

12

09/491694
01/26/00

-1-

Also enclosed are:

- ___ A Declaration.
- ___ An Assignment and Recordation Form Cover Sheet.
- ___ A certified copy of a priority application.
- ___ A Power of Attorney.
- ___ A Statement Claiming Small Entity Status.
- ___ An Information Disclosure Statement under 37 C.F.R. § 1.156.

The filing fee pursuant to 37 C.F.R. § 1.16 is determined as follows:

No. Filed	No. Extra	Rate Small Entity/ Other Than Small Entity		
Basic Fee		\$345.00 \$690.00	=	\$
Total Claims ___ - 20	= ___* X	\$ 9.00 \$ 18.00	=	\$
Independent Claims ___ - 3	= ___* X	\$ 39.00 \$ 78.00	=	\$
First Presentation of Multiple Dependent Claim(s) ___		\$130.00 \$260.00	=	\$
		Total	=	\$

*If the difference is less than zero, enter "0".

- ___ Please charge Deposit Account No. 06-1325 in the amount of \$___. A duplicate copy of this authorization is enclosed.
- ___ A check in the amount of \$___ to cover the filing fee (\$___), and assignment recording fee (\$40.00), if applicable, is enclosed.
- ___ The Commissioner is hereby authorized to charge underpayment of any additional fees (including those listed below) or credit any overpayment associated with this communication to Deposit Account No. 06-1325. A duplicate copy of this authorization is enclosed.
- ___ Any additional filing fees under 37 C.F.R. § 1.16.

— Any patent application processing fees under 37 C.F.R. §1.17.

This application is filed pursuant to 37 C.F.R. §1.53(b) in the name of the above-identified inventor.

— This application claims priority to an earlier-filed Provisional patent application, as set forth more fully in this application.

Please direct all correspondence concerning the above-identified application to the following address:

Larry E. Vierra
FLIESLER, DUBB, MEYER & LOVEJOY LLP
Four Embarcadero Center, Suite 400
San Francisco, California 94111-4156
Telephone: (415) 362-3800

Respectfully submitted,

Date:

Jan. 26, 2000

By:

Larry E. Vierra
Larry E. Vierra
Reg. No. 33,809

FLIESLER, DUBB, MEYER & LOVEJOY LLP
Four Embarcadero Center, Suite 400
San Francisco, California 94111-4156
Telephone: (415) 362-3800

DATA TRANSFER AND SYNCHRONIZATION SYSTEM

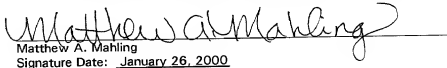
INVENTOR

David L. Multer

**CERTIFICATE OF MAILING BY "EXPRESS MAIL"
UNDER 37 C.F.R. § 1.10**

"Express Mail" mailing label number: **EL 498 238 645 US**
Date of Mailing: January 26, 2000

I hereby certify that this correspondence is being deposited with the United States Postal Service, utilizing the "Express Mail Post Office to Addressee" service addressed to Box **PATENT APPLICATION**, Assistant Commissioner for Patents, Washington, D.C. 20231 and mailed on the above Date of Mailing with the above "Express Mail" mailing label number.


Matthew A. Mahling

Signature Date: January 26, 2000

DATA TRANSFER AND SYNCHRONIZATION SYSTEM

INVENTOR:

David L. Multer

Limited Copyright Waiver

5 A portion of the disclosure of this patent document contains material to which the claim of copyright protection is made. The copyright owner has no objection to the facsimile reproduction by any person of the patent document or the patent disclosure, as it appears in the U.S. Patent and Trademark Office file or records, but reserves all other rights whatsoever

BACKGROUND OF THE INVENTION

10 Field of the Invention

The invention relates to the transference of data between two systems independent of the form in which the data is kept on the respective systems, and in particular to providing an efficient means of communicating data between systems and devices.

15

Description of the Related Art

20 The growth of computing-related devices has not been limited to personal computers or work stations. The number of personal computing devices has grown substantially in both type and format. Small, hand-held computers carry a multitude of contact, personal, document, and other information and are sophisticated enough to allow a user to fax, send e-mails, and communicate in other ways wirelessly. Even advanced cellular phones carry enough memory and processing power to store contact information, surf the web, and provide text messaging. Along with the
25 growth in the sophistication of these devices, the need to transfer

information between them has grown significantly as well.

5 With a multitude of different device types on the market, keeping information between the different devices synchronized has become increasingly problematic. For example, if an individual keeps a calendar of information on a personal computer in his or her office using a particular personal information manager application, the individual would generally like to have the same information available in a cellular phone, hand-held organizer, and perhaps a home personal computer. The individual may additionally have a notebook computer which requires synchronizing file 10 data such as presentations or working documents between the notebook and the office computer.

Until now, synchronization between both documents and personal information managers has occurred through direct connection between the devices, and generally directly between applications such as a personal 15 information manager in one device and a personal information manager in another device or using an intermediary sync-mapping program.

One example of this is the prevalent use of the 3Com Palm® OS-based organizer, such as the 3Com Palm® series of computing devices, which uses its own calendaring system, yet lets users synchronize the data 20 therein with a variety of different personal information manager software packages, such as Symantec's ACT!™, Microsoft's Outlook®, and other systems. In this example, an intermediary synchronization program such as Puma Technology, Inc.'s Intellisync® is required. Intellisync® is an application program which runs on both the hand-held device and the 25 computer which stores the information data and maps data systems between non-uniform data records. In other cases, direct transfer between applications such as transfer between Microsoft's Outlook® computer-based client and Microsoft's Windows CE "Pocket Outlook" application, is possible. Nevertheless, in both cases, synchronization occurs through 30 direct connection between a personal computer and the personal

computing device. While this connection is generally via a cable directly connecting, for example, Palm® device in a cradle to the personal computer, the connection may be wireless as well.

5 One component of these synchronization systems is that the synchronization process must be able to delineate between when changes are made to specific databases and must make a decision about whether to replace the changed field. Normally, this is measured by a change in one database, and no-change in a second database. In some cases, both
10 databases will have changed between syncs. In this case, the sync operation must determine which of the two changes which has been made is to "win" and replace the other during the sync. Generally, this determinant of whether a conflict exists allows some means for letting the user resolve the conflict.

15 In a technical sense, synchronization in this manner is generally accomplished by the copying of full records between systems. At some level, a user is generally required to map data fields from one application to another and specify which data fields are assigned to which corresponding field in a different device. Less mapping is required where developers more robustly support various platforms of applications.

20 In many instances, the data to be synchronized is generally in the form of text data such as records of addresses, contact information, calendar information, notes and other types of contact information. In certain instances, data to be synchronized will be binary format of executable files or word processor-specific documents. In many cases
25 where document synchronization is required, the synchronization routine simply determines whether or not the documents in question have changed, and uses a time-based representation to determine which of the two files is newer, and replaces the older file with the newer file to achieve synchronization, as long as the older of the two files was in fact not
30 changed. This is the model used in the familiar "Briefcase" function in

Microsoft Windows-based systems. If both files have changed, then the synchronization routine presents the option of conflict resolution to the user.

5 Such synchronization schemes are generally relatively inefficient since they require full band-width of the document or binary file to be transferred via the synchronization link. In addition, at some level the synchronization programs require interaction by the user to map certain fields between different programs.

10 One of the difficulties in providing synchronization between different computing devices is that the applications and platforms are somewhat diverse.

15 Nevertheless, all synchronization programs generally require certain functions in order to be viable for widespread usage. In particular, synchronization programs must work with popular applications on various platforms. Sync applications must allow for conflicts resolution when changes are made to the same information on different devices between syncing events. They must provide synchronization for all types of formats of data, whether it be text data in the form of contacts, e-mails, calendar information, memos or other documents, or binary data in the form of documents or programs in particular types of formats.

20 In a broader sense, applications which efficiently synchronize data between disparate types of devices can provide advantages in applications beyond synchronizing individual, personal information between, for example, a personal information manager hardware device such as a
25 Palm® computing device, and a personal computer. The same objectives which are prevalent in developing data transfer between personal information management (PIM) devices and desktop systems lend themselves to furthering applications requiring data transfer between other types of devices, on differing platforms. These objectives include speed,
30 low bandwidth, accuracy, and platform independence.

For example, current e-mail systems use a system which is somewhat akin to the synchronization methods used for disparate devices in that an entire message or file is transferred as a whole between different systems. When a user replies to an e-mail, generally the entire text of the original message is returned to the sender, who now has two copies of the e-mail text he/she originally sent out. The same is true if an e-mail attachment is modified and returned. All of the text which is the same between both systems is essentially duplicated on the originator's system.

SUMMARY OF THE INVENTION

The invention comprises a system and method for efficiently, quickly and easily synchronizing devices which can couple to the Internet, or any network. Synchronization of the devices can occur at independent times using an intervening network based storage server to store changes to data for all the different devices in the system in a data independent format, and provide the data on request to the device requesting a sync at the time requested. Hence, two devices need not be coupled to each other to perform a sync.

In one aspect the invention comprises a system for synchronizing data between a first system and a second system. The system includes a first sync engine on the first system interfacing with data on the first system to provide difference information. A data store is coupled to network and in communication with the first and second systems. A second sync engine is provided on the second system coupled to receive the difference information from the data store via the network, and interfacing with data on the second system to update said data on the second system with said difference information.

Difference information is transmitted to the data store by the first sync engine and received from the data store from the second sync engine. The difference information is transmitted to the data store at a first

point in time, and received from the data store at a second, subsequent point in time. In a further aspect, the second sync engine can interface with said data on the second system to provide second difference information to the data store and the first sync engine may thereafter couple to the data store to retrieve the second difference information and interface with the data on the first system to update said data on the first system with said second difference information.

The system may include a management server coupled to the network and in communication with the first sync engine, the second sync engine and the data store.

In a further aspect, the system may include a first device, coupled to the first system via the network, providing said data to the first system. In such instance, the first system may be a sync server.

The system may include a plurality of sync engines on a respective plurality of systems, each of said plurality of engines being coupled to receive difference information from each of said first, second and plurality of sync engines from the data store via the network. Each said engine interfaces with data on the system on which it resides to update said data on said system on which it resides with said difference information, and interfaces with data on said system on which it resides to provide difference data information from the system on which it resides to the data store.

In a further embodiment, the invention comprises a system including a first device, a data store and a second device. The first device includes at least a first data file and first differencing code having an input and an output coupled to a network to receive first device data change transactions, based on said at least one data file, from and provide change transactions to, said network. The data store is coupled to the network and has at least one data structure coupled to store change transactions. The second device includes at least a second data file and second

differencing code having an input and an output coupled to the network to receive said first device data change transactions, and provide second change transactions based on said at least second data file to said data store.

5 In a further embodiment, the invention comprises a method for synchronizing at least a first and a second resident on a first and a second systems, respectively, coupled to the Internet, respectively. The method includes the steps of: determining difference data resulting from changes to the first file on the first system; transmitting the difference data to a server via the Internet; querying the server from a second system to determine whether difference data exists for files on the second system; 10 retrieving the difference data to the second system; and updating the second file on the second system with the difference data.

15 In a still further aspect, the invention comprises an Internet synchronization system. The system includes a storage server having an Internet connection; a first device coupled to the Internet and including a device sync engine; and a second device coupled to the Internet and including a second device sync engine. A management server may further be provided. In this aspect, each device sync engine may comprise an 20 application object, an application object store, and a delta engine.

BRIEF DESCRIPTION OF THE DRAWINGS

25 The invention will be described with respect to the particular embodiments thereof. Other objects, features, and advantages of the invention will become apparent with reference to the specification and drawings in which:

Figures 1 - 7 are block diagrams of various configurations of the system of the present invention utilizing the differencing routines of the present invention.

30 Figure 8 is an overview of one embodiment of the system

architecture in accordance with the present invention.

Figure 9A is a block diagram of the desktop device engine of the present invention.

5 Figure 9B is a block diagram of the configuration of server side device engines utilized in accordance with the present invention.

Figure 10 is a block diagram of one embodiment of the device engine in an operating system such as Windows.

Figure 11 is a block diagram of an application object incorporated into the device engine of the present invention.

10 Figure 12 is a diagram of storage object hierarchy of a universal data format utilized with the system of the present invention.

Figure 13 is a listing of exemplary item objects used in accordance with the routines of the present invention.

15 Figure 14 is a block diagram of a management storage server architecture for used in the system of the present invention.

Figure 15 is a flow diagram illustrating a pull synchronization in accordance with the system of the present invention.

Figure 16 is a flow diagram illustrating a push synchronization in accordance with the system of the present invention.

20 Figure 17 is a diagram of the management server architecture in accordance with the present invention.

DETAILED DESCRIPTION

25 The present invention includes a system and a method for transferring data between two devices which require information to be shared between them. In accordance with the discussion herein, a "device" is defined as a collection of elements or components organized for a common purpose, and may include hardware components of a computer system, personal information devices, hand-held computers,
30 notebooks, or any combination of hardware which may include a processor

and memory which is adapted to receive or provide information to another device; or any software containing such information residing on a single collection of hardware or on different collections of hardware. Such software might include applications such as personal information managers, which include contact data and other such information, e-mail systems, and file systems, such as those utilized by Microsoft Windows NT operating systems, Unix operating systems, Linux operating systems, or other systems capable of storing file types having binary formats which translate to application formats of differing types.

5 In one embodiment, the invention comprises a set of programs specifically designed to transmit and/or receive differencing data from one device to another device, irrespective of the type of file system, data, content, or system hardware configuration.

10 In a further aspect, the system comprises store and forward technology which utilizes the differencing technology to implement services via a public or private network, such as the Internet.

15 The system of the present invention finds particular usages in synchronizing personal contact information between different systems, but it will be readily apparent to one of average skill in the art that the present invention provides advantages having broader applicability than merely synchronizing various types of systems. For example, replying and forwarding e-mail can be made more efficient by forwarding only the differences in e-mails between systems. As a further example, updates to systems software via a network can be made more efficient where, for
20 example, instead of completely replacing different modules of an application, only the differences of the modules need be forwarded, resulting in more efficient use of existing bandwidth.

System Overview

25 Figures 1-7 show various configuration alternatives of the present

invention.

Figure 1 shows an embodiment of the present invention in a basic configuration. In Figure 1, a first system or device, system A, and a second system or device, system B, are coupled by a communication line 110. It should be readily understood that communication line may be any direct coupling of the two systems which allows data to pass between the systems such as, for example, by means of serial ports, parallel ports, an Ethernet connection or other type of network, or an infrared link, or the like. System A includes a functional block 100 representing a differencing transmitter in accordance with the present invention. System B includes a functional block 102 representing the differencing receiver in accordance with the present invention.

The differencing transmitter 100, upon receipt of a control signal enabling operation of the transmitter, examines a specified data structure of information which is to be transmitted to system B. Differencing transmitter 100 extracts such information from System A and converts the information extracted into difference information Δ . Difference information Δ comprises only the changes to System B's data which have occurred on System B and instructions for implementing those changes. Hence, if the data to be transferred is a change to a file which exists on system B, difference information Δ comprises only the differences in such file and where such differences occur. If the data does not exist at all on System B, the difference information Δ will be the entire file. Difference information Δ received by differencing receiver 102 at System B is reconstructed at System B, and the changes reflected therein are updated on System B.

For example, if System A and System B are two computers and an update for certain binary files on System A is required, the differencing transmitter on System A will extract the differences in the file known to exist on System B and any new files, and transmit only those differences (an instructions for where to insert those differences) to the differencing

receiver 102. Differencing receiver 102 will interpret the difference information (Δ) and reconstruct the binary files on System B. In this manner, the information on System B is updated without the need to transfer the entire binary files between the Systems.

Figure 2 shows a second example of the system of the present invention. In Figure 2, both System A and System B include functional blocks 104, each representing a differencing synchronizer. The function of the synchronizer 104 is similar to that of the transmitter and receiver combined; the synchronizer will allow difference information Δ to be both transmitted and received. For example, System A and System B are a portable computer and a desktop computer, respectively, where information such as contact information needs to be synchronized between the two, the differencing synchronizer 104 will extract changes made to the contact information on either System A or System B and at predetermined times, transmit the information Δ between the systems, and reconstruct the data on the receiving system to update information from the sending system, in order to ensure that both systems contain the same data.

Figure 3 shows yet another alternative embodiment of the system of the present invention. In Figure 3, System A again includes a differencing transmitter and System B includes a differencing receiver 102. In this embodiment, a storage server 300 is coupled between System A and System B. Storage server 300 may store a separate database of the difference information Δ provided by System A, which allows System A to provide its difference information Δ to the storage server 300 at a first point in time, and storage server 300 to provide the same difference information Δ to System B at a second point in time, but not the same as the first point in time. In addition, multiple sets of difference information Δ may be provided at different points in time, and stored for later retrieval by System B. Still further, the difference information sets may be maintained on server 300 to allow data on either System A or System B to be returned to

a previous state.

Once again, the storage server 300 is coupled by a direct connection 110 to both System A and System B. Storage server 300 may be a server specifically adapted to receive differencing information Δ from the receiver 100 and provide it to the transmitter 102. In one embodiment, server 300 includes specific functional routines for enabling this transfer. Alternatively, server 300 comprises standard information server types which respond to standard Internet communication protocols such as file transfer protocol (FTP), or hypertext transfer protocol (HTTP).

Figure 4 shows yet another alternative embodiment of the system of the present invention wherein System A and System B, once again coupled directly to a storage server 300 by a direct connection line 110, each include a differencing synchronizer 104. Difference information Δ can be passed to and from System A through synchronizer 104 to and from the storage server 300 at a first point in time, and to and from System B at a second point in time. In this embodiment, storage server 300 may include routines, described below, for resolving conflicts between data which has changed on both System A and System B independently after the last point in times when the systems were synchronized.

Figure 5 shows yet another alternative embodiment of the present invention including four systems: System A which includes a differencing synchronizer 104; System B which includes a differencing receiver 102; System C which also includes a differencing synchronizer 104; and System D which includes a differencing transmitter 100. Each is directly coupled to a storage server 300, allowing control of transmission of differencing data Δ between the various systems. Server 300 may include routines, described in further detail below, to track the various types of systems which comprise System A through System D, and which control the transmission of various components of the difference information Δ to each of the various systems. For example, since System B includes only

5 differencing receiver 102, the difference information Δ_2 which is provided to it may be a sub-component of that which is transferred between System A in the storage server 300, or may be simply receiving broadcast information Δ_4 from System D. In one embodiment of the system of the present invention, server 300 does not itself route the difference information derived from each receiver/transmitter/synchronizer. Server 300 acts as a repository for the information, and the determination of which difference information Δ is attributed to which receiver/transmitter/synchronizer is made by each receiver/transmitter/synchronizer.

10 Figure 6 shows yet another alternative embodiment of the present invention. In Figure 6, a synchronizer is provided in storage server 300. It should be recognized that a forwarder and/or receiver may be provided in server 300 as well. The particular embodiment shown herein may be advantageous where device processing power and memory are limited, such as cases where the device is a cell phone. It should be noted that the data transferred between system A and the device engine 104a in such an embodiment may or may not be difference information, depending on whether System A has the capacity to detect and output difference information. Each of the devices may include a differencing receiver, a differencing transmitter, or a differencing synchronizer. It should be understood that a portion of the differencing synchronizer 104a may reside on System A and another portion may reside on server 300.

20 Figure 7 shows yet another alternative embodiment of the present invention wherein the devices shown in Figure 6 may be coupled to a combination of public or private networks 700 such as, for example, the Internet. The network 700 may include one or more storage servers 300, 300₂, and in such cases the difference information Δ transmitted between each such device 602-610 via intermediate storage on one of such servers. Network 700 may couple the devices to one or more specialized function servers, such as servers specifically designed to

include a differencing forwarder, receiver or synchronizer. Such devices may comprise, by way of example and without limitation, a personal office PC 602, a smart telephone 604, a user's office PC 606, a personal information Palm® computing device 608, a telephone or cellular phone 604, a home personal computer 606, or a web browser 610. Each differencing receiver, differencing transmitter, or differencing synchronizer present in devices 602-610 includes means to poll the data stored on storage servers 300₁,300₂ to determine whether the data present at storage server 300₁,300₂ includes difference information which the particular receiver or synchronizer is required to have to synchronize the data on the device on which it resides.

In the following description, an embodiment wherein the differencing receiver, transmitter, and synchronizer are described will be discussed with respect to its use in synchronizing contact information, calendar information, and binary file information between a plurality of different devices in the context of data synchronization. It will be readily understood that the system of the present invention is not limited to synchronization applications, or applications dependent upon specific types of data, such as contact information or scheduling information. In particular, it will be readily understood that the transmission of data comprising only the differences in data between two systems via routines which extract the data and reassemble data on the various systems, represents a significant advancement in the efficient transmission of data. The present invention allows for optimization in terms of a reduction in the bandwidth utilized to transmit data between two systems, since only changes to data are transferred. This consequently increases the speed at which such transactions can take place since the data which needs to be transmitted is substantially smaller than it would be were entire files transferred between the systems.

In a particular embodiment of the present invention, the ability of

devices to connect to the Internet is leveraged to manage data transfer between the systems. In essence, each particular device which requires information access which can connect to the Internet may become part of the system of the present invention, and synchronize its data with other devices defined by a user as being part of the system.

Generally, the system comprises client software which provides the functions of the differencing transmitter 100, differencing receiver 102, and differencing synchronizer 104 in the form of a device engine. The device engine includes at least one component particular to the type of device on which the device engine runs, which enables extraction of information from the device and conversion of the information to difference information, and transmission of the difference information to the storage server. This allows the replication of information across all systems coupled to the system of the present invention. Although the storage servers 300 utilized in the system of the present invention may be any type of storage server, such as an Internet server or an FTP server, and may be provided from any source, such as any Internet service provider (ISP), particular aspects of a storage server which may be useful and which may be customized to optimize transfer of information between systems coupled as part of the present invention will be described below. Synchronization of devices utilizing the synchronization system of the present invention is possible as long as an Internet connection between the devices is available.

In a key aspect of the invention, the Internet connection between the devices or between the devices and a server, need not exist at the same point in time, and new devices may be added to the system of the present invention at any point in time without the loss of information. The system provides totally transparent access to information and the device engine on each device provides an operating system independent extension which allows seamless integration of the personal information services in accordance with the present invention.

In a particular unique aspect of the present invention, only those changes to the information which are required to be forwarded to other systems on the system of the present invention are transmitted to enable exceptionally fast response times. In a still further aspect of the invention, information which is transferred in this manner is encrypted to ensure security over the public portions of the Internet.

Architecture Overview

Figure 8 shows an overview of the architecture of the system of the present invention utilized for synchronizing or "syncing" information on different types of devices. In the embodiment hereinafter described, the system of the present invention allows the coupling of a collection of personal devices and applications one uses when working with personal information. Nevertheless, the system may be used to broadcast public or private information to various device types. System software in the form of a device engine for each device which is declared a part of the system of the invention is distributed across the collection of devices to enable synchronization. Distribution of the device engines may occur via, for example, an installation package forwarded over an Internet connection. In essence, the device engine software of the present invention forms a distributed processing network which maintains consummate synchronization of all information in the system. The processing load associated with delivering this service is pushed to the end-point devices which provides for easy scaling of the system to ever-larger applications.

The present invention contemplates the use of two types of device engine: one totally embodied on the server which outputs change data to the server; and a second totally embodied on the server receiving device generated change information from the device. In addition, a hybrid of the two, having a portion of the device engine on the device and a portion on the server, is disclosed.

As shown in Figure 8, any number and type of devices 802-808 may be utilized in accordance with the system of the present invention. A telephone 802 may comprise a cellular phone or a standard POTS-connected telephone. Telephone 802 may include contact information and, as is supported with a newer generation of cellular telephones, appointments and task data stored in a data structure 812. The application 812 which utilizes the application data 822 comprising such information is all stored in the telephone unit 802. Likewise, a personal digital assistant such as a Palm® computing device 804 includes application 814 and application data 824 which may include information such as contacts, appointments and tasks, and may also include file information such as documents which are created and stored on the PDA 804. Device 806 is represented as a Windows personal computer running an operating system such as Microsoft Windows 95, 98, NT or 2000. Applications 816 which may be running on device 806 include the Windows operating system itself, Microsoft Outlook, Symantec's ACT Personal Information Manager, Goldmine Software's Goldmine, Lotus Organizer, Microsoft's Internet Explorer web browser, Netscape's Communicator Suite, Qualcomm's Eudora e-mail, and various other programs, each of which has its own set of application data 826 which is required to be synchronized not only with devices outside the system 806, but also between devices and applications within the system itself. Finally, a dedicated web browser client 808 is shown which couples via the Internet to web portal applications 816 which have their own set of application data 828. Unlike devices 806 which store the application and application data substantially in their own hardware, web portal applications are provided on a separate server and provided to browser 808 via an Internet connection. Nevertheless, the web portal application stored on the portal application provider includes a set of application data 828 which a user may wish to synchronize. For example, a large web

portal such as Yahoo! and Snap.com provide services such as free e-mail and contact storage to their users. A user may wish to synchronize this with applications running on their cellular phone, PDA, or Windows devices.

5 In order to access the specific application data of each of the systems shown in Figure 8, a device engine is associated with each type of device. A cellular device engine 862 communicates and incorporates itself with the application data 822 of the cellular phone. Likewise, a PDA device engine 864 is provided, which may be based on either the Palm®
10 operating system, Windows CE operating system, or other PDA-type operating systems as necessary. A Windows-based device engine 866 includes a mechanism, discussed below, for extracting application data 826 from supported Windows applications 816, and a web services device engine 868 incorporates to extract application data 828 from web portal
15 applications 818.

As shown in Figure 8, some device engines are provided entirely on the device (and are referred to herein as desktop device engines), while others include components at the back end server (which may comprise storage server 850 or a specialized server, as shown in Figure 9B.) This
20 is illustrated generally by lines 832, 834, 836, and 838 in Figure 8. Also, in Figure 8, elements above dashed line 855 are provided by an administrator or service provider of the system of the present invention. Each of the device engines 862, 864, 866 and 868 is configured relative to the type of device on which it resides. For example, the Cell phone
25 device engine 862 includes one or more components arranged on the phone while others are on server 850. Conversely, device engine 866 resides entirely on the windows device 806.

Data from each of the devices is coupled via an Internet connection 710 with a storage server 850. As noted above, storage server 850 may
30 be a generic storage server or it may be a storage server specifically

adapted for use with the system of the present invention as discussed below. One or more of the storage servers 850 are used to communicate transactions amongst the collection of systems 802, 804, 806, 808. It should be readily recognized that any number of different types of systems 802, 804, 806, 808 may be provided in accordance with the present invention and incorporated into the system. However, for brevity, not all the different types of commercially available computing devices which are currently in use or in development, in which the system of the present invention may be incorporated, are listed.

In its simplest embodiment, the storage server 850 is simply a dumb storage server and each of the device engines transmits only difference information thereto to be stored in a particular location accessible by other device engines in the system. In one embodiment, each device engine implements all processing required to keep all the systems fully synchronized. Only one device engine needs to be coupled to the storage server 850 at one particular point in time. This permits synchronization of multiple systems in a disconnected fashion. Each device engine will download all transactions encapsulating changes that have occurred since the last synchronization from the server and apply them to the particular device.

The change or difference information (Δ) is provided in one or more data packages, the structure of which is described herein. Each data package describes changes to any and all transfer information across all device engines, including but not limited to application data, files, folders, application settings, and the like. Each device engine can control the download of data packages that include classes of information that apply to the specified local device 802, 804, 806 or 808 attached to that specific device engine. For example, device engine 862 will only need to work with changes to information describing contact names and phone numbers in application data 822, while device engine 866 will be required to work with

changes to e-mail, changes to document files, notes, as well as contact and address information since the application data 826 is much more extensive than application data 822.

Each device engine includes compression/decompression and encryption/decryption components which allow encryption and/or compression of the data packages transmitted across Internet connection 710. It should be recognized that compression and encryption of the data packages may be optionally provided. It is not required in accordance with the present invention. Each device engine performs mapping and translation steps necessary for applying the data packages to the local format required for that type of information in the application data stores 822-828. The device engine also includes components which allow it to track ambiguous updates in cases where users have changed data to a particular data field on two different systems simultaneously since the last update. In this case, the device engine includes a mechanism for drawing this to the attention of the user and allowing the user to resolve the conflict.

Device Engine Architecture

Figure 9A illustrates a single device engine utilized with a generic application 810 and a generic storage server 850. Figure 9A illustrates a desktop device engine, since all processing occurs on the device and only difference information is transmitted to server 850. Nevertheless, an understanding of the desktop device engine will aid in understanding server side devices engines, hereinafter described. Shown in Figure 9 are the functional components of a device engine in block form and their interrelationship to each other. The device engine 860 is equivalent to the functional block of a differencing sequencer 104 shown in Figures 1-7.

While the invention will be described with respect to the embodiment of the invention as a differencing synchronizer 104, it will be

readily understood that portions of the functionality are utilized as needed in a forward-only (a differencing transmitter) or a receive-only (a differencing receiver) capacity as required by the particular application.

As noted above, a device engine exists for each and every device that makes up a user's personal information network of devices in the system. As shown in Figure 9A, each device engine 860 includes an application object 910. The application object is specific to each particular application 810 and provides a standard interface between the device engine and the balance of the data transmission system of the invention, and the application 810. Details of the application object will be described in further detail below. The application object is a pluggable architecture which supports a wide variety of vendor-unique applications. The job of the application object is to map data from the application into a temporary or "universal" data structure by connecting to the application via any number of standard interfaces to gain access to the applications data. The data structure of the application object puts the data in a generic or "universal data" format which may be used by the device engine components to generate data packages for provision to the storage server.

Also provided is an application object store (AOS) 920 which includes a copy of the device's data at a point just after the previous data extraction and synchronization occurred. Application object store 920 is a mirrored interface which stores a snapshot of the previous state of the data from the application object 910 in the device engine. The size of the AOS will depend on the data being collected by each device engine.

The generic output of the application object is provided to a delta module 950. Delta module 950 is a differencing engine which calculates differences in data between the output of the application object 910 and the copy of the data which is provided in an application object store (AOS) 920. The actual differencing and patch routine can comprise a routine such as XDelta or YDelta. The delta module 950 will be referred to herein

alternatively in certain portions of the description as "CStructuredDelta."
In addition, the difference information is alternatively referred to herein as
a "change log." Each change log (or set of difference information) is a self
describing series of sync transactions. As described below, the change
log may be encrypted and compressed before output to the network.

Hence, during a sync, the Application Object will, using a
mechanism discussed below, extract the data of each application in the
device and convert it to a universal data format. The delta module will
then generate a difference set by comparing the output of the Application
Object and the AOS. This difference information is forwarded to the
encryption and compression routines for output to the storage server 850
in the form of a data package. Alternatively, the data from one application
can be used to synchronize to data in another application in, for example,
a windows environment, as shown by arrow 1050 in Figure 10.

It should be specifically noted that the application object may
interface directly unstructured binary data or with structured application
data. The differencing routine supports both uses of the delta module 950
in comparison generation.

In some cases, operation of the application object and delta module
is simplified by the fact that some applications, such as PDA's, have the
ability to output changes to its data. In such cases, the delta module 950
need only provide the data into the data package, since comparison to an
AOS is not required – the application already includes a mechanism for
tracking changes made to its own data. However, in many cases the
applications provide, at most, a standard interface to access the data, such
as Microsoft's ODBC interface, the Microsoft standard Application
Programming Interface (API), or other similar standard interfaces.

Device engine 860 further includes a versioning module which
applies a version number per object in the data package. As explained
further below, each object in the data package is assigned a universally

unique ID (UUID). Hence, unlike many prior synchronization systems, the system of the present invention does not sync data solely by comparing time stamps of two sets of data. Versioning module 915 allows each device engine to check the state of the last synchronization against data packs which have been provided to the storage server to determine which data packages to apply. This allows the device engine to sync itself independently of the number of times another device engine uploads changes to the storage server. In other words, a first device engine does not care how many times a second device engine uploads data packages to the server.

An events module 925 controls synchronization initialization events. Items such as when to sync, how to sync, trigger the delta module 950 to perform a synchronization operation.

A user interface 930 is provided to allow additional functional features to a system user of the particular device to which the device engine 860 is coupled. The user interface is coupled to a conflict resolution module 940, a filtering module 945, and a field mapping module 935. Each of the modules provides the functionality both necessary for all synchronization programs, and which users have come to expect.

Filtering module 945 allows filtering for types of content based on, for example, a field level content search. The field mapping module 935 allows for the user to re-map certain interpretations of items which were provided in the document stream. For example, if the device engine 860 is operating on a personal computer, and a synchronization is occurring between the personal computer and a notebook computer, and the user has a "my documents" directory on the personal computer which he wishes to map to a different directory on the notebook computer, the field mapping module 935 allows for this re-mapping to occur. It should be recognized that the field mapping module allows for changes in directing the output of the data package. The field mapping module 935 is not necessary to map

particular data fields of, for example, contact information from one application, such as Microsoft Outlook, to a different application, such as Symantec's ACT, as is the traditional use of field mapping and synchronizing applications.

5 Delta module 950 is further coupled to a compression module 970 and an encryption module 960. It should be recognized that the compression encryption modules need not be enabled. Any type of compression module 970, such as the popular PK Zip or Winzip modules, or those available from HiFn Corporation may be utilized in accordance
10 with the invention. Moreover, any type of encryption algorithms, such as MD5, RCH 6, Two Fish, or Blowfish, or any other symmetric encryption algorithm, may be utilized. In one embodiment of the invention, encryption without compression is used. In a second embodiment of the invention, compression without encryption is used. In a third embodiment of the
15 invention, neither compression or encryption is used, and in a fourth embodiment of the invention, both compression and encryption are used.

 Versioning module 915 also allows the device engine 860 to support multiple users with distinct synchronization profiles. This allows multiple users accessing the same machine to each synchronize their own data set
20 using the same device engine. For example, if the application 810 on a particular device comprises Microsoft Outlook on a personal computer, coupled to a Microsoft Exchange server, and Outlook is configured to have multiple user profiles, versioning module 915 will track the data applied through the device engine when a sync request occurs. This allows two
25 users of the same Outlook client software which access different data sets, either in the client computer or on a separate server, to utilize the same device engine and the system of the present invention via the same machine. In a further embodiment, a particular device engine supports the use of foreign devices accessing the system via the same connection.
30 Palm® devices, for example, use a cradle to connect to a computer and/or

Internet connection. If a particular user wishes to allow another user to use his Palm® pilot cradle connection to synchronize the other user's Palm® pilot, the device engine can generate data packages to update the local application object store for the foreign device. The application object store can therefore be used as a temporary storage for cases allowing synchronization of foreign devices.

The output of the device engine 900 comprises a data package which is output to storage server 850. As noted above, only one device engine need be connected to the storage server 850 at a given time. The data package can be stored on the storage server 850 until a request is made to a particular location of the storage server by another device engine. Likewise, delta engine 900 can query alternative locations on the storage server for access to synchronized data within the system of the present invention. Access to areas of the storage server is controlled by a management server (MS) described more fully below. In one embodiment, each sync operation requires that the device engine for each device login to the management server to authenticate the device and provide the device engine with the location of the individual device's data packages on the storage server.

Data packages may be advantageously provided to the device engine from the storage server in a streaming format, allowing processing to occur using a minimum of bandwidth and storage in the devices. The device engine 860 and particularly the delta module 950 interpret data packages based on the versioning information and the mirrored data present in the application object store 920. When data is returned to the delta module 950 from the storage server 850, the delta module returns differenced data to the application object 910 for the particular application which then translates the delta information into the particular interface utilized for application 810. Once a device engine has been fully applied all data packages from an input stream, it generates a series of data

packages that describe the changes made on the local system. The device engine uses the local application object store 920 to keep track of the last synchronized version of each application's actual data, which is then used for the next data comparison by the delta module on the next sync request. Generated data packages can include operations and encode changes generated from resolving ambiguous cases as described above.

Figure 9B depicts how server based device engines may be provided in the system of the present invention. The Palm® device example is shown in this embodiment, where the Palm® device has the capability of connecting directly to the Internet and a service provider's data center 900. The data center includes a firewall 975 to prevent unauthorized communications with servers resident in the data center 900 and protect integrity of the data. The storage server 850 may communicate directly through the firewall as may the management server (MS) 1410.

Shown therein are two sync servers 982 and 984 each of which is dedicated to syncing one particular type of application. Sync server 982 is dedicated to the Palm® device, while sync server 980 is dedicated to, for example, a portal application (Portal1).

Since the Palm® Device 804a includes a mechanism for transmitting changes to its data directly, data may be transmitted using HTTP request and response via the firewall 975 to the sync server 982 where differencing and updating of data in the AOS can occur, after which changes can be downloaded to the Palm® 804a.

The synchronization server is an application handles concurrent synchronization of user's data. Each Sync Server includes plug-in support for multiple devices to be synchronized using the same sync server executable. Each device type has it's own device name that identifies which AO / AOS components will be used during the sync.

5 The sync server uses the concept of a universal data record in its internal sync differencing engine and when sending data to and retrieving from external entities such as the AOS and AO. Hence, in the Palm® application, the job of a server AO is simply to take the device-specific format of its record and convert into a universal record format.

10 The Sync Server has a plug-in architecture so that 3rd party application partners can easily add their services into the server. Currently, if the server is operated in a Microsoft Windows NT Server, the sync server discovers the sync components via the Windows NT registry. In alternative embodiments, this function is performed in a Component
15 Manager which operates on each sync server to manage processing by each of the AO and AOS on the server. Each AO and AOS are implemented as a stand-alone DLL that the Sync Server loads at initialization time, or when adding a new component via the Component Manager.

Each sync server is shown as dedicated to a single application. However, a sync server may handle multiple device types.

20 In the embodiment of Figure 9B, it should be noted that, depending on the device type, there are different configurations for the AOS and AO's. For example, the Palm®'s AO data store 1050 resides on the Palm® device 804a itself and a separate AOS data store 1052 exists for this configuration (an Oracle database). In the case of Portal1, the AOS and AO use the data store 1054.

25 Device engines can generate additional data packages intended to resolve synchronization problems in other systems. For example, interfacing with the conflict resolution module 940, if the user makes a change to a particular data store on an application object on his Palm® pilot, then makes an additional change to a personal information manager (PIM) application on his personal computer, the user can specify that the
30 change made on the personal computer will "win" when the conflict is

detected by the Δ engine and the versioning information between the two devices. This is essentially a definition that one particular set of data is correct and should replace the second set of data.

Figure 10 shows a specific embodiment of a desktop device engine
5 utilized in, for example, a Microsoft Windows-based operating system environment.

As shown in Figure 10, a Windows operating system may have at least three specific applications which may require synchronization. In Figure 10, the system includes Netscape Communicator application 1040
10 having data such as bookmarks 1021, contacts 1022, and e-mail 1023; a Microsoft Outlook application 1042 which includes contact information 1024, calendar information 1025, e-mail information 1026, note information 1027, and tasks information 1028; and Windows operating system 1044
15 information including Favorites data 1029, file system information 1030, and individual files 1031.

Each particular application 1040, 1042, 1044 has an associated application object 1010, 1012, 1014. Each of the respective application objects provides data back to delta module 950 in a generic format which is usable by the delta module in accordance with the foregoing description
20 of the apparatus shown in Figure 9A. From Figure 10, it will be additionally seen how the delta module 950 may be utilized to synchronize data between applications running on the same particular server. The device engine hence does an intra-system sync such as, for example, between the contact information 1022 from Netscape and the contact
25 information 1024 from Outlook.

Figure 10 further illustrates the modularity of the system of the present invention allowing the device engine to include any number of different application objects to be provided on a single device to incorporate all applications run on that device.

30 In operation, during an installation of a device engine into a

particular system, the installation program may be tailored to provide application objects which may be present on a given system. For example, and with reference to Figure 10, the installation program for a Windows machine will carry any number of application objects for systems and applications which may be present on a Windows machine. The installer will check for the presence of given applications, and allow the user to add additional applications which may be installed in locations that are not the normal default installation areas for application support by the application objects which the installer is carrying, or de-select certain applications which, for one reason or another, the user may not wish to install an application object for and render a part of the system of the present invention.

Application Object Structure

Figure 11 is a conceptual depiction of the structure of an application object. As noted above, the application object is a pluggable architecture which supports a wide variety of vendor-unique applications. The consistent and scalable architecture of the system of the present invention for device engines is maintained by encapsulating system-dependent knowledge in a single component, i.e. the application object. As noted above, every application object supports a standard set of interfaces that every device engine understands. Each application object maps these standard interfaces of the capabilities of a particular vendor application. Hence, there will be as many application objects as there are application types.

As noted above, there are different types of server and desktop device engines, some having application objects entirely on the server, while others have application objects entirely on the desktop.

Each application object will include a connector 1110 which may comprise a generic interface to the particular application for which the

application object store has been designed. For example, when connecting to a Palm® device, the connector will be an HTTP protocol request routine which interfaces with the Palm® device's own built-in synchronization manager, which provides an output of records which have been changed on the Palm® device. As in Figure 9B, since the Palm® outputs all the changes to its data via its own sync manager, in the Palm® application, the job of a server AO is simply to take the device-specific format of its record and convert into a universal record format.

The connector provides access for the application object to remove the data field from a particular application and convert it to a universal record structure. In the desktop AO, where, for example the application object is designed for a Windows interface, the connector may be the Windows API and the job of the AO will be to translate data from, for example, the windows file system to a universal data format. This universal data structure is then used by the delta module 950 to build data packages to be used in synchronization between components of the systems provided in the network system of the present invention.

Universal data structure mapping, used on desktop application objects, and universal data record mapping, used by the server device engines, is further detailed below.

Desktop Application Object

Each Application Object (AO) is a software component that interfaces with the third party application APIs (Application Programming Interface) to provide the programming services to the delta module for extraction and deposition of information data from and to the third party application domain during synchronization. In addition, the AO maps the third party application data fields to system's domain.

The AO service is a collection of COM (Component Object Model) objects that can be developed in conjunction with the third party Windows

application APIs as a form of a DLL (Dynamic Linked Library) in C or C++. The DLL is loaded on demand at runtime during synchronization. It should be recognized that the application object need not be implemented using the COM model, but may be developed with other distributed object models.

There are a number of the related subsystems and documents that the developer must be familiar with and this document has made many references to those subsystems during the course of presenting the AO.

- *Change Log (CL)* (or differencing information) , a data file which contains a series of synchronization transactions.
- *DataPack*, a compacted and encrypted Change Log.
- *StructuredDelta*, the delta module differentiation engine that generates differences between Application Objects and Change Log and AOS.
- *AOS*, a database which resides locally on, for example, a windows machine.
- *MS*, a management server that manages users' accounts.
- *SS*, an FTP or storage server that manages data packs.
- *User Manager*, a standalone Windows client UI program that manages the synchronization process.
- *ePortal*, a web-based PIM portal site.
- *pio_types.h*, a header file which contains the definitions of the system's supported data fields known as tags.
- *Def.h*, a header file contains the definitions of the system's constants.
- *interfaces.h*, a COM interface file contains AO interface definitions.

Each AO has a COM interface-based design built-in. That is, instead of providing a set of traditional APIs as programming services, it provides a set of interface-based objects as programming services.

StructuredDelta, the delta module, the primary intended user of

each AO. StructuredDelta instantiates these COM objects and uses them throughout the synchronization session exclusively through the COM interfaces on those objects to interface with the third party application database.

- 5 Each AO component consists of a set of objects that translate the third party application data into the universal data middle format which underpins the entire spectrum of PIM data regardless of which third-party application the data comes from. The objects in universal data format are **device, (application) data class, store, folder, item, and data fields.**
- 10 The AO digests the third party application data of any kind and reduces it into a few handful simple objects and field types. These objects and field types are fed into StructuredDelta engine and are compared by StructuredDelta in order of their hierarchy. The resulting differences (add, delete, modify) are logged as transactions in the difference information.
- 15 The data packs are transported to a storage server that may be actively managed by a management server for each individual user account and devices.
- 20 StructuredDelta uses AO objects to access and modify the individual AO objects and data fields. AO objects serve as a buffer between individual application data and StructuredDelta so that StructuredDelta does not require knowledge of each application and database. All AO objects are temporary and created in the space of each AO by StructuredDelta through COM interfaces. AO objects are referenced when they are in use and they are freed when StructuredDelta stops using them. One can think of AO objects as merely placeholders of each application objects for StructuredDelta to access. Once StructuredDelta has a particular Application's data, StructuredDelta would free AO objects immediately without storing them internally.
- 25
- 30

AppObj

AppObj is a root object of each AO component and there is one and only one per AO. AppObj provides an entry point into the individual application's database. StructuredDelta instantiates it and holds it on during the entire synchronization session and releases it afterward. AppObj offers a number of services such as what class of data it supports. The C++ example of AppObj's definition is shown below:

```

class CMyFIAppObj :
10     public    Item,
        public    AppObj,
        protected ModuleIdentity,
        protected DataClassInfo,
        protected ItemTypeInfo,
        protected ItemFieldMap,
        protected FolderInfo,
        protected DataFileInfo,
        protected SynchNotify,
        protected ErrorMsg,
        protected EnumItems,
        protected FindItem,
        protected ModifyItem
20
    {
    public:
25        CMyAppObj( HWND hWndParent );
        ~CMyFppObj();
    };

```

AppObj can contain children objects. They are Store objects. **EnumItems** interface is used to enumerate Store objects. **FindItem** interface is used to find the contained objects. **ModifyItem** interface enables AppObj to create a new Store object. AppObj is created by StructuredDelta calling *CreateAppObject(HWND hWndParent, AppObj*
***pObj)*.

Store

The Store object represents a database of the individual application information. If the individual application can handle multiple databases at same time, one needs multiple Store objects. One can think of Store object
as a specialized Folder object, the root folder of each particular application

data domain. The C++ example of Store's definition is shown below:

```

class CMyStore :
    public Item,
    public ItemContainer,
5    protected EnumItems,
    protected FindItem,
    protected FindItemByData,
    protected ModifyItem,
    protected ReadWrite
10 {
    CMyStore();
    ~CMyStore();
};

```

Store is a container of Folder objects. **EnumItems** interface enables the enumeration of its contained folders while **FindItem** and **FindItemByData** interface is used to find contained Folders or Item objects. **ModifyItem** and **ReadWrite** interface enables the modification of each application database.

Folder

Folder object is a specific data class of each individual application such as a table in the relational database or a collection of data records in each application. For example, the applications contact collection can be thought as a Folder object. The C++ example of Folder's definition is shown below:

```

class CMyFolder :
    public Item,
    public ItemContainer,
30    protected EnumItems,
    protected FindItem,
    protected FindItemByData,
    protected ModifyItem,
    protected ReadWrite
35 {
    public:
        CMyFolder();
        ~CMyFolder();
40 };

```

Folder object is also container. It can contain Item objects as well as Folder objects. **EnumItem** interface allows the enumeration of either Folder objects or Item objects or both. **FindItem** and **FindItemByData**

interface is used to find contained Folder objects or Item objects. **ModifyItem** and **ReadWrite** interface enables the modification of an application's data tables.

Item

Item object represents an individual entity of each application's domain specific data. Item object can be thought as a record of each application's relational table. For example, a contact, email, calendar, to-do item in the particular application can be thought of as an Item object.

The C++ example of Item's definition is shown below:

```
class CMyItem :
    public Item,
    protected EnumItems,
    protected FindItem,
    protected ModifyItem,
    protected ReadWrite
{
    public:
        CMyItem();
        ~CMyItem();
};
```

Item can contain Attachment objects only. EnumItems interface enables the enumeration of Attachment objects if any. **ModifyItem** and **ReadWrite** interface enables the modification of an application's records or data fields.

Attachment

Attachment object is a specialized Item object that encapsulates an attachment data or relationship. Only Item can have Attachment objects. Attachment object can be thought as attachment data such as attached-email files. Attachment can also be thought as attachment relationship to other Item objects. The example of that is the distribution list (Item object) can contain contacts (Item objects). The C++ example of Item's definition is shown below:

```
class CMyItemAttachment :
```

```
public Item,
protected ReadWrite,
protected ModifyItem

{
    public:
        CMyItemAttachment();
        ~CMyItemAttachment();
};
```

Variant

Variant object represents a data field of each particular application data. For example, a 'first name' of a contact or the birthday date of a contact can be thought as Variant object. StructuredDelta only understands Variant object and the types of data fields it encapsulated.

Variant object can contain any one of the following data field type:

```
struct Variant
{
    enumFieldTag      tag;
    enumFieldDataFlag flag;    // flags item fields as not
    known or otherwise special
    union
    {
        short int      i;        // eFieldType_WORD
        LONG            l;        // eFieldType_LONG
        DWORD           dw;       // eFieldType_DWORD
        unsigned__int64 qw;       // eFieldType_QWORD
        UUID            uuid;     // eFieldType_UUID
        DATE            time;     // eFieldType_DATE
        LPTSTR          psz;      //
        eFieldType_String
        Binary          bin;      // eFieldType_Binary
        Float          flt;       // eFieldType_Float
        Double          dbl;      //
        eFieldType_Double
        FIcollection    coll;     //
        eFieldType_Collection
        ) Value;
        Stream*         strm;     // eFieldType_Stream
    };
};
```

Variant::tag is an identification tag of data field and variant::flag specifies the type of data field while Variant::value member variable stores each application's field value. One data field type is **Collection**. **Collection** object is an array of Variant objects. It can be used to represent a compound data fields.

```
struct Collection
{
    ULONG            cValues;
    struct_Variant** paVar;    // This array really
```



```
contains cValues entries
};
```

- 5 Another data field type that is worth exploring is Binary. Binary object can be used to represent a binary data as it is.

```
struct Binary
{
10       ULONG       cb;
      LPBYTE       lpb;
};
```

15 AO Interfaces

Each AO object has an AO COM interface. Each object must implement some of those interfaces to create certain capability or desired behavior that are expected by StructuredDelta.

20 IItem

- This is the base interface of all application objects. It provides the identification service to StructuredDelta. Every object must have a unique ID, parent unique ID, display name, and item type information (*eltemType_FOLDER*, *eltemType_CONTACT*, etc). The unique ID is a unique string only in a given device. It is not persistent cross the Internet to other devices. The ID usually comes from the third party application database domain such a unique ID of a record.

```
interface IItem : Unknown
{
30       STDMETHOD_(LPCTSTR, GetUniqueID) () const PURE;
      STDMETHOD_(LPCTSTR, GetParentUniqueID) () const PURE;
      STDMETHOD_(LPCTSTR, GetDisplayName) () const PURE;
      STDMETHOD_(enumItemType, GetItemType) () const PURE;
35       STDMETHOD_(BOOL, IsContainer) () const PURE;
      STDMETHOD_(DATE, GetLastModificationTime) () const PURE;
      STDMETHOD_(QWORD, GetSize) () const PURE;
      STDMETHOD_(DWORD, GetFlags) () const PURE;
} ;
```

40 IItemContainer

This is the base interface of all application container objects (**store, folder**). These container objects must have this interface implemented so

that StructuredDelta would recursively descend in them if they have IItemContainer capability.

```

5 interface IItemContainer : IItem
{
    STDMETHOD_(BOOL, ContainsItemType) ( enumItemType eItemType ) PURE;
    STDMETHOD_(BOOL, ContainsDataClass) ( enumDataClass eDataClass )
PURE;
    STDMETHOD_(enumSpecialFolderType, GetSpecialFolderType) () PURE;
10 };
    STDMETHOD_(GUID, GetMappingGUID) () PURE;

```

IErrorMsg

15 This is an error-reporting interface for every application object. It is used by StructuredDelta to query the error string after a failure. The AO should implement this on every object after the error occurs and before returning the control to StructuredDelta.

```

20 interface IErrorMsg : IUnknown
{
    STDMETHOD(GetErrorString) ( LPTSTR pszError, int iBufLen
) const PURE;
};

```

IEnumItems

25 This is an interface for collection enumeration, used by StructuredDelta to enumerate the objects of the third party application database. **IItemEnumFlags** (eItemEnumFlags_FOLDER, eItemEnumFlags_ITEM, and eItemEnumFlags_ATTACHMENT) is used to
30 enumerate only the requested type of objects.

```

interface IEnumItems : IUnknown
{
    STDMETHOD(ItemQueryStart) ( enumItemType type, long
35   &lCount, eItemEnumFlags dwFlags ) PURE;
    STDMETHOD(ItemQueryNext) ( Item **ppItem ) PURE;
    STDMETHOD(ItemQueryFinish) () PURE;
};

```

IFindItem

40 This is an interface for recursively finding object within the third party application database, used by StructuredDelta to find application object by its unique ID.

```

interface IFindItem : IUnknown
{
    STDMETHOD(FindStoreByID) ( LPCTSTR pszUniqueID, ItemContainer
        **ppFolder ) PURE;
    STDMETHOD(FindFolderByID) ( LPCTSTR pszUniqueID, ItemContainer
        **ppFolder ) PURE;
    STDMETHOD(FindItemByID) ( LPCTSTR pszUniqueID, Item **ppItem )
        PURE;
};

```

IFindItemByData

This is an interface for recursively finding the object that matches the search criteria data. The search criteria are represented as **Collection** that allows the multiple search field keys to be used during the search. The multiple objects may be found that match the search criteria. The interface also provides enumeration capability of the search results.

```

interface IFindItemByData : IUnknown
{
    STDMETHOD(FindByDataStart) ( enumItemType type, Variant*
        pSearchKey, int* pnFound ) PURE;
    STDMETHOD(FindByDataNext) ( LPTSTR pszEntryID, int
        cbBufSize ) PURE;
    STDMETHOD(FindByDataFinish) () PURE;
};

```

IModifyItem

This is an interface for StructuredDelta to add, delete, and re-parent application data in the third party database during synchronization.

```

interface IModifyItem : IUnknown
{
    STDMETHOD(Add) ( BOOL bFolder, enumItemType type, Item
        **ppItem ) PURE;
    STDMETHOD(Delete) () PURE;
    STDMETHOD(Move) ( ItemContainer * pDestFolder ) PURE;
};

```

IReadWrite

This is an interface for accessing, writing, and mapping the third party application data fields by StructuredDelta. It provides the capability of read and write data fields from and to the third party application database and the capability of mapping data field of the third party application to universal data format of the system of the present invention.

Any object that has data fields and require field level synchronization must implement this interface.

```
5      interface IReadWrite : IUnknown
      {
          STDMETHOD(Read) () PURE;
          STDMETHOD(Commit) () PURE;
          STDMETHOD(GetFieldData) ( enumFieldTag fieldTag, Variant
10      **ppVariant ) PURE;
          STDMETHOD(ReleaseFieldData) ( Variant *pVariant ) PURE;
          STDMETHOD(SetFieldData) ( const Variant *pVariant )
          PURE;
      };
```

15

IAppObj

This is an AppObj only interface. It provides the capability of logon and logoff to the third party applications during synchronization. The data class filter mechanism is used by StructuredDelta to filter the enumeration of contained data classes (eDataClass_CONTACT, eDataClass_CALENDAR, etc).

```
20      interface IAppObj : IUnknown
      {
          STDMETHOD(Logon) ( HWND hWndParent ) PURE;
          STDMETHOD(Logoff) () PURE;
          STDMETHOD(SetFilter) ( const VOID* pFilter, int BufLen )
          PURE;
          STDMETHOD (int, GetFilter) ( VOID* pFilter, int BufLen )
          PURE;
30      };
};
```

25

30

IModuleIdentity

This is an AppObj only interface. It provides DLL module identification information to the Manager object such as the name of the third party application, enum ID of this application, and the application installation detection support.

```
40      interface IModuleIdentity : IUnknown
      {
          STDMETHOD(GetName) ( LPCTSTR pszName, int iBufLen ) const
          PURE;
          STDMETHOD(GetAppl) ( Appl *pAppl ) const PURE;
          STDMETHOD(IsInstalled) ( BOOL *bisInstalled ) const
          PURE;
45      };
};
```

IItemTypeInfo

This is an AppObj only interface. It provides the information on the number of item types supported by AO, what type items are supported and the capabilities for a specific item type. This returns a DWORD containing bits set.

```
interface IItemTypeInfo : IUnknown
{
    STDMETHOD(GetSupportedTypesCount) ( int &iCount ) PURE;
    STDMETHOD(GetSupportedTypeInfo) ( int iIndex,
enumItemType &type, LPTSTR pszTypeName, int iBufLen ) PURE;
    STDMETHOD(GetItemTypeCaps) ( enumItemType type, DWORD
&dwFlags ) PURE;
};
```

IDataClassInfo

This is a CAppObj only interface. It provides the information on the number of data classes that are supported by the application object and what the data classes are supported

```
interface IDataClassInfo : IUnknown
{
    STDMETHOD(GetCount) ( int *piCount ) PURE;
    STDMETHOD(GetDataClass) ( int iIndex, enumDataClass
*peDataClass ) PURE;
};
```

IDataFileInfo

This is a CAppObj only interface, it provides information on the number of database files and database filenames supported by AO to avoid being synched twice by application sync and file-set sync.

```
interface IDataFileInfo : IUnknown
{
    STDMETHOD(GetDataFileCount) ( int *piCount ) PURE;
    STDMETHOD(GetDataFilePath) ( int iIndex, LPTSTR
pszFilePath, int iBufLen ) PURE;
};
```

IItemFieldMap

This is a CAppObj only interface that is used by StructuredDelta to query the data fields of given application object. For example, what are

data fields in application object called eltemType_CONTACT?

```
5      interface IItemFieldMap : IUnknown
      {
          STDMETHOD(FieldQueryStart) ( const enumItemType &type,
          int &iCount ) PURE;
          STDMETHOD(FieldQueryNext) ( enumFieldTag &field, LPTSTR
          pszName, int iBufLen, LPTSTR pszType, int iTypeBufLen )PURE;
          STDMETHOD(FieldQueryFinish) ( ) PURE;
      };
```

10

IFolderInfo

This is a CAppObj only interface, used by StructuredDelta to obtain the special and default folders' unique IDs and UUIDs.

```
15      interface IFolderInfo : IUnknown
      {
          STDMETHOD(GetSpecialFolderID) ( enumSpecialFolderType
          eFolder, LPTSTR pszUniqueID, int iBufLen ) PURE;
          STDMETHOD(GetDefaultFolderID) ( enumItemType type,
          LPTSTR pszUniqueID, int iBufLen ) PURE;
          STDMETHOD(MapFolderGUID) ( UUID uuidFolder, LPTSTR
          pszUniqueID, int iBufLen ) PURE;
      };
```

25

IFastSync

This is a CAppObj only interface that is used by StructuredDelta to query if the given AO also provides FastSync service or not. FastSync is a DLL component that is written using the third party APIs and loaded into the third party application to receive the changes in database while users are operating the application. It is used to speed up the synchronization performance by syncing only the objects that are known to IFastSync component.

```
35      interface IFastSync : IUnknown
      {
          STDMETHOD(GetFastSync) ( enumDataClass eDataClass, BOOL*
          pbFastSync ) PURE;
      };
```

40

SynchNotify

This is a CAppObj only interface that is called by Manager to notify the third party application the state of synchronization: **start**, **finished**, or

reset so that the application can prepare itself accordingly.

```
interface ISynchNotify : IUnknown
{
    STDMETHOD(SynchNotify) ( enumSynchNotify eNotify ) PURE;
};
```

Server AO

Server Application Objects share many characteristics with desktop application objects, including support for reading and mapping to the universal record structure set forth above.

Nevertheless, among various devices incorporated into the system of the present invention, each application object database will be quite different. For example, the Palm® database on the device is really just a memory space with records laid out sequentially in memory. In a web portal-type application, the application object may be an Oracle database. Server application objects may generally have less difficult tasks since the applications supported are generally either devices providing their own change data output, (such as Palm®-type PDA's), or which do not have a great deal of data to export (such as cell phones, having only name and number information).

Nevertheless, each application object must support all calls defined in a class interface definition as follows:

FUNCTION	DESCRIPTION
Open	Perform an initialization of the device before data retrieval functions are called.
Close	Done with database calls, cleanup if necessary.
Get First Modified Record	Get the first modified record from the device and insert into application object.
Get Next Modified Record	Get the next modified record from the device and insert into the application object.

Add Record	Add a record into the application object database.
Update Record	Update a record.
Delete Record	Delete a record in the application object database.
Set Device Records	A function called during the synchronization manager to send a bytestream to the application object for interpretation. The bytestream will contain a list of records to add to the application object modified records list. At a later point in time, such records will be retrieved by the Get First Modified Record/Get Next Modified Record functions.
Get Device Records	For records bound to the device, this call gets a bytestream that contains a list of records to add back to the device. There is an outbound record list that is saved until this call is finished, at which time the sync server will be finished with the application object.
Set Device Response	A function used to modify or repair a record input saved in the application object store that was sent to the device in the Get Device Records call, such as a record ID for a record. If 10 records were sent to the device during the Get Device Records call, one would expect to see 10 records coming back in during this function call.

As noted above, because each application object database is different, the calling convention and the application object itself will likewise be different. The calling convention for a Palm® device's sync manager application object is given in the following pseudo-code:

```

Call AO::Open
Call AO::WriteRecords

Start synchronization process

While more records in AO Data Object
    Call AO::GetFirstModifiedRecord()
    Call AO::GetNextModifiedRecord()
END

IF new records THEN
    Call AO::AddRecord()
IF deleted records THEN
    Call AO::DeleteRecord()

```



```
IF update record THEN
    CALL AO::UpdateRecord()

Call AO::Close
```

As shown therein, the calling convention is designed to be integrated with the Palm's® own sync manager.

A second example provided below shows mapping of a portion of a web portal's contact database:

```
MappingItem CContactTable::m_FieldMap[] =
{
    {1, eFieldTag_Contact_FirstName, "firstname"},
    {1, eFieldTag_Contact_MiddleName, "middlename"},
    {1, eFieldTag_Contact_LastName, "lastname"},
    {1, eFieldTag_Contact_Title, "title"},
    {1, eFieldTag_Contact_Suffix, "suffix"},
    {1, eFieldTag_Contact_Anniversary, "anniversary"},
    {1, eFieldTag_Contact_Birthday, "birthday"},
    {1, eFieldTag_Contact_AssistantName, "assistantname"},
    {1, eFieldTag_Contact_Children, "children"},
    {1, eFieldTag_Contact_CompanyName, "companyname"},
    {1, eFieldTag_Contact_Department, "department"},
    {1, eFieldTag_Contact_FTPSite, "ftpsite"},
    {1, eFieldTag_Contact_Gender, "gender"},
    {1, eFieldTag_Contact_JobTitle, "jobtitle"},
    {1, eFieldTag_Contact_ManagerName, "managername"},
    {1, eFieldTag_Contact_NickName, "nickname"},
    {1, eFieldTag_Contact_Office, "office"},
    {1, eFieldTag_Contact_Profession, "profession"},
    {1, eFieldTag_Contact_Spouse, "spouse"},
    {1, eFieldTag_Contact_SelectedMailingAddress, "selectedmailingaddress"}
};

int CContactTable::m_nNumFields =
sizeof(m_FieldMap)/sizeof(MappingItem);

HRESULT CPortalAddrOCI::InsertRecord( MappingItem theMap[], int
numFields, CDataAccessor *pInsertItem, CFIItemUniversal *pUnivItem,
bool bForceCreate )
{
    bool bHasData = SetRecordFields( theMap, numFields,
pInsertItem,
pUnivItem );

    if( bHasData || bForceCreate )
    {
        // Insert the record into the database and execute the command
        pInsertItem->InsertRow(0);
        pInsertItem->Exec();
    }

    return S_OK;
}
```

The above example of mapping the contact field files maps contact fields from a particular web contact information database to fields in the universal record format from the master list header file (pio_types.h) in the

system of the present invention. This mapping is for a specific contact table and it should be understood that other information, such as phone numbers, e-mail addresses, and other contact information may be stored in a separate table.

5 Once data is extracted from a particular application, the server application object must then convert the information into the universal record format which can be utilized by other server device engines to take content information into their own particular application.

10 Universal Record Format

 The universal record format is used by each server device engine to handle various tasks of encapsulating records in a common format, comparing records, creating and holding differences between records, and other tasks of synchronization.

15 The universal record format allows the application objects to support a wide range of extensible application item types such as contacts, calendar, mail, bookmarks, and the like. Flexible type name and value associations permit synchronization without regard to individual vendor application information formats. Each application object encapsulates mapped knowledge from the vendor unique format to the universal format
20 of the present invention. As such, an application object can be designed to support any combination of application and binary information types. In essence, application objects can be designed to support a vendor application using only binary file synchronization if the internal format of
25 the application is not known.

 Server application objects can also be designed to create collections. For example, if the user wishes to create a "my pictures" collection which consists of some collection of information and synchronize this collection of information, such an arbitrary grouping of classes of
30 information into appropriate representations is supported.

Because the connector layer of the interfaces to the actual storage with a vendor application varies with application type, application access methods can include, but are not limited to, disk or database access, network protocols, wireless device protocols, and the like.

- 5 The Universal Records Format and the Universal Field Format class definitions are given below:

```

10 typedef map < enumFieldTag, CUniversalField, less_enumFieldTag >
   UniversalRecordMap;
   typedef UniversalRecordMap::value_type UniversalRecordPair;
   typedef UniversalRecordMap::iterator UniversalRecordIterator;
   typedef UniversalRecordMap::const_iterator
   ConstUniversalRecordIterator;

15 class CUniversalRecord
   {
   private:
       UniversalRecordMap recordMap_;
   public:
20     bool conflicts(const CUniversalRecord& rhs);
       bool add(const CUniversalRecord &rhs);
       bool subtract(const CUniversalRecord& rhs);

       CUniversalRecord();
       CUniversalRecord( const CUniversalRecord& rhs );
       virtual ~CUniversalRecord();

       // add this element
30     HRESULT insert( enumFieldTag eId, long value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
       HRESULT insert( enumFieldTag eId, LPCWSTR value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
       HRESULT insert( enumFieldTag eId, DATE value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
35     HRESULT insert( enumFieldTag eId, string value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
       HRESULT insert( UniversalRecordPair p );

       CUniversalRecord exclusiveLeftWins( CUniversalRecord& rhs );
       CUniversalRecord inclusiveLeftWins( CUniversalRecord& rhs );
       bool removeSame(const CUniversalRecord &rhs);

       bool Find( const enumFieldTag eId, CUniversalField &field );
45     UniversalRecordMap::iterator find( const enumFieldTag eId ) {
       return recordMap_.find(eId); }

       UniversalRecordMap::iterator begin()           { return
       recordMap_.begin(); }
       UniversalRecordMap::iterator end()             { return
50     recordMap_.end(); }
       bool empty()                                   { return
       recordMap_.empty(); }
       long size()                                    { return
55     recordMap_.size(); }
       UniversalRecordMap::iterator erase(UniversalRecordMap::iterator
       it)
       { return recordMap_.erase(it); }
       void clear()                                   { recordMap_.clear(); }
   };

```

The UniversalField Structure

```

5  class CUniversalField
   {
   public:
       enum eUniversalField
       {
10          eUF_Unknown,
            eUF_Long,
            eUF_String,
            eUF_Date,
            eUF_Blob
15      };

   protected:
       eUniversalField      typeId_;
       enumFieldTag         fieldId_;
       enumFieldDataFlag    flag_;
       size_t               len_;

       union
       {
20          long      l_;
            DATE      d_;
            TCHAR*   pCh;
       } value_;

30  public:
       CUniversalField();
       CUniversalField( const CUniversalField& rhs );
       CUniversalField( enumFieldTag itemId, long value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
35      CUniversalField( enumFieldTag itemId, DATE value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
       CUniversalField( enumFieldTag itemId, LPCTSTR value,
       enumFieldDataFlag flag = eFieldDataFlag_Normal);
       CUniversalField( enumFieldTag itemId, string blob,
40      enumFieldDataFlag flag = eFieldDataFlag_Normal);
       ~CUniversalField();

       bool operator==( const CUniversalField& rhs ) const;
       bool operator!=( const CUniversalField& rhs ) const { return
45          !operator==(rhs); }
       CUniversalField& operator=( const CUniversalField& rhs);

       eUniversalField getType() const { return typeId_; }
       enumFieldTag getFieldID() const { return fieldId_; }
50      enumFieldDataFlag getFlag() const { return flag_; }
       size_t getLength() const { return len_; }

       LPCTSTR getString() const { ASSERT( eUF_String ==
       typeId_); return value_.pCh; }
55      long getLong() const { ASSERT( eUF_Long ==
       typeId_); return value_.l; }
       DATE getDate() const { ASSERT( eUF_Date ==
       typeId_); return value_.d; }
       string getBlob() const { ASSERT( eUF_Blob ==
60      typeId_); return string(value_.pCh,len_); }

       void get( LPCTSTR& p ) const { ASSERT( eUF_String ==
       typeId_); p = value_.pCh; }
       void get( long& l ) const { ASSERT( eUF_Long ==
65      typeId_); l = value_.l; }

```

```

void get( DATE& d) const          { ASSERT( eUF_Date ==
typeId_); d = value_.d; }
void get( string& b) const{ ASSERT( eUF_Blob ==
typeId_); b.assign(value_.pCh,len_); }

5      bool isString() const      { return typeId_ ==
eUF_String; }
      bool isLong() const        { return typeId_ == eUF_Long; }
10     bool isDate() const        { return typeId_ == eUF_Date; }
      bool isBlob() const        { return typeId_ == eUF_Blob; }
};

```

Example

An example of how data is removed from one particular application data type and converted into the universal record format is given below for an Oracle database:

```

20     #include "stdafx.h"

#include <string>
using namespace std;
#include "FlItemUniversal.h"

25     #include "oci.h"
#include "OCIDefs.h"
#include "OCIConnect.h"
#include "OCISession.h"
30     #include "OCIColumn.h"
#include "OCICursor.h"
#include "DataAccessor.h"

#include "UniversalMapper.h"
35     #include "UniversalRecord.h"
#include "FlUtil.h"

#include "BaseAOSTableOCI.h"

40

/*
 * Function: MapFields
 * Description: Map fields from an Oracle database record into an
45     UniversalRecord format.
 */

void CBaseAOSTableOCI::MapFields( CDataAccessor *pAccessor, MappingItem
50     theMap[], int numFields, CUniversalRecord &univRec )
{
    string sValue;
    DATE      dtValue;
    LONG       lValue;
    double     dValue;

55     for( int inx=0; inx<numFields; inx++ )
    {
        enumFieldTag fieldID = theMap[inx].m_universalFieldID;

60         switch( FlPropType( fieldID ) )

```

```

(
    case eFieldType_Binary:
    {
5      // to fill properly, 1st name and last name should already be assigned
        CUniversalField emailField;
        string sValue;

        if ( SUCCEEDED( BuildEmailField( pAccessor,
10      sValue, emailField ) ) )
            univRec.insert( fieldID,
            emailField.getBlob() );
        break;
    }
15      case eFieldType_String:
        if( pAccessor->GetFieldValue( fieldID, sValue
        ) )
        {
20          if ( 0 == ::wcslen(sValue.c_str()) )
              continue;

          univRec.insert( fieldID, sValue.c_str()
25      );
        }
        break;

        case eFieldType_DATE:
            if( pAccessor->GetFieldValue( fieldID, dtValue
30      ) )
                univRec.insert( fieldID, dtValue );
            break;

        case eFieldType_DWORD:
            if( pAccessor->GetFieldValue( fieldID, lValue
35      ) )
                univRec.insert( fieldID, lValue );
            break;

        case eFieldType_Double:
40      if( pAccessor->GetFieldValue( fieldID, dValue
        ) )
            univRec.insert( fieldID, dValue );
            break;
45      }
    }
}

50  HRESULT CBaseAOSTableOCI::InsertRecord( MappingItem theMap[], int
    numFields, CDataAccessor *pInsertItem, CFlitemUniversal *pUnivItem, bool
    bForceCreate )
    {
55      bool bHasData = SetRecordFields( theMap, numFields, pInsertItem,
        pUnivItem );

        if( bHasData || bForceCreate )
        {
60          pInsertItem->InsertRow(0);
          pInsertItem->Exec();
        }

        return S_OK;
65      }
    /*

```

```

* Function: SetRecordFields
* Description: Map fields from an UniversalRecord format into an
Oracle database record (pInsertItem)
*/
5  bool CBaseAOSTableOCI::SetRecordFields( MappingItem theMap[], int
numFields, CDataAccessor *pInsertItem, CPlItemUniversal *pUnivItem )
{
    bool bHasData = false;
    CUniversalField field;
10     for( int inx=0; inx<numFields; inx++ )
    {
        enumFieldTag fieldID = theMap[inx].m_universalFieldID;
15         BOOL bExists = pUnivItem->m_record.Find( fieldID, field );
        if( bExists )
        {
            bHasData = true;
            if( field.isBlob() )
            {
                string blob = field.getBlob();
                LPCTSTR szEmailAddr =
25         GetAddressFromRecipients( (FIRECIPIENTS*)blob.c_str() );
                if( szEmailAddr && *szEmailAddr != NULL )
                    pInsertItem->SetFieldValue( fieldID,
                    (string)szEmailAddr );
            }
            else
            {
                bHasData = false;
                continue;
            }
        }
        else if( field.isString() )
35         {
            string sValue = field.getString();
            if( !sValue.empty() )
                pInsertItem->SetFieldValue( fieldID,
                sValue );
40         }
        else if( field.isLong() )
        {
            LONG nValue = field.getLong();
            pInsertItem->SetFieldValue( fieldID, nValue );
45         }
        else if( field.isDate() )
        {
            DATE dValue = field.getDate();
            if( dValue )
50             pInsertItem->SetFieldValue( fieldID,
            dValue );
        }
    } // if( bExists )
55     } // For all fields
    return bHasData;
}
60

```

While the above-identified code is specific to, for example, an Oracle database, one of average skill in the art will readily recognize that the

technique utilized above may be adapted to other types of databases containing records and fields of interest. In the above code examples, all fields which are mapped from a particular application are mapped to fields in the master mapping file.

5

Management Server

In order to provide security and identification of particular users in an Internet-implemented synchronization system, a management server may be provided in the system of the present invention. The management server is a centralized server which controls behavior and characteristics of the entire network of device engines across all users.

Figure 14 shows the general representation of how a management server 1410 integrates itself into the system of the present invention. Shown in Figure 14 is an exemplary device engine 1450 which has HTTP links to both a management server 1410, a storage server 1415, and a generic FTP server 1420. As will be discussed hereinafter with reference to the process of the present invention, and the specific implementation of the data below shown in Figures 15-17, the management server interacts with the device engine to control authorized access to information on the storage server, or a generic FTP server 1420, 1425 to access device-specific information storage 1430 in accordance with the system of the present invention. This allows any device coupling to the Internet to have access to management protocols and to retain user information across all platforms which the data which is being synched by the system of the present invention must access.

The management server communicates using hypertext transfer protocol (HTTP) which may be implemented with a secure sockets layer (SSL) to ensure security.

In particular, the management server supports an authentication interface that requires each device engine to authenticate with the management server before performing synchronization. Certain storage

server implementations may utilize locking semantics to control read and write access to storage for multiple device engines. For example, in a generic FTP request, if two device engines attempt to connect to the same data at the same time, there must be some form of locking control to prevent device engines accessing the same data at the same time. In this instance, the management server controls the device engine acquisition, renewal, and releasing of locks against data stored in the network.

Each device engine is uniquely identified and tracked by the management server. This allows for tailoring behavior between the management server and specific types of storage systems and device engine components. All device engine components are tagged and version stamped for management via the management server.

Device actions can request updated copies of individual device engine components, permitting self-update and configuration of device engine systems. This permits minimal download designs for device engines that are on low bandwidth connections enabling the device engines to download additional required components at a later time.

In a further aspect of the system, a value added component may be provided where the management server can support client's advertising mechanisms, enabling the display of banner or similar advertising on a device engine system without the need for a web browser. Cycling of advertisements, statistic collection, and the like, are managed via management server protocols. Online purchase and subscription mechanisms are also supported using the management server protocol.

The management server further supports the accounting, sign-up registration, device edition, storage server selection, and similar functions for each user in the system. In one embodiment, the management server may retain password and encryption information for a given user account. In a second embodiment, such information is not retained. The second embodiment provides the advantage that users may feel more secure if the

maintainer of the management server is not in possession of the password to access data in the user's account.

Further information with respect to the management server and the data flow from the management server to other components of the system of the present invention will become apparent with respect to the discussion of the process flow and data flow diagrams in Figures 15-17.

Figure 17 shows a general depiction of the data flow and the functional specification of the management server utilized in accordance with the present invention.

As shown in Figure 17, following a welcome request 1710, a user is allowed to sign out which enables an add user module 1712, and subsequently enables an add device module 1714. If sign-up is not requested, information may be provided via module 1718.

As indicated in Figure 17, the add user module 1712 adds user records to the user in device database 1750. Additionally, the add device module 1714 adds users and devices to the user device database 1750. A device list 1720, and a device engine download and update database 1722, provide selection data for the add device module 1714. The account authentication module 1724 receives input both directly from a user log-in from the welcome screen at 1710 and from the add device module 1714.

Once an account is authenticated and confirmed, the administrator of the system of the present invention having a private data store at 1770 may choose to provide a web desktop 1754 which allows access to a user's records such as file 1756, e-mail 1758, calendar 1760, contacts 1762, notes 1764, and tasks 1766. The information will be culled from a provider database 1752 which will be synched in accordance with the system of the present invention as previously described. In essence, the provider database 1752 accesses data from the device engines 1780, which include, as discussed above, the storage server, each individual device engine 1785, and a settings database 1787.

Other portions of the management server include the locking modules for beginning a sync 1732, continuing a sync 1734, and ending a sync 1736, and for updating user information including modifying a user 1742, adding devices 1744, removing devices 1746, and modifying devices 1748.

Storage Server

Shown in Figure 14 is the storage server 1415. While storage server 1415 may include a generic storage model accessible through any number of standard Internet protocols, in accordance with the present invention, a flexible storage architecture is provided that permits various standard implementations of the system of the present invention. This allows deployment of network services without installation of new server applications and can be responsible for communicating change information between multiple device engines in a consistent fashion.

One or more storage servers 1415 may be used to communicate transaction amongst a collection of devices. Each user's personal information network is represented by a unique account within its own data package storage section. The storage server 1415 maintains persistent store collection of data packages which is, at a minimum, enough data packages to be capable of synchronizing the most out-of-date system in a user's given information network or add information to new devices which are provided in the network. Additional data packages can be maintained to permit rollback of previous versions of information. The storage server can automatically dispose of older data package storage and can support aging of an inactive accounts.

Each storage server 1415 may be implemented using a variety of implementations including a standard FTP server for any operating system platform. The storage server can be implemented using HTTP protocols for increased efficiency and firewall avoidance. The storage server may be implemented using techniques for local storage such as database access or

single file storage of a user's entire file system tree. The storage server 1415 may utilize the stored foreign protocol model for moving copies of data packages to other storage servers in the system. In one embodiment, the storage server can allow tunneling of information using an alternative protocol to other storage servers in cases where firewall prevents originating protocol. For example, a storage server can relay an FTP traffic inside an HTTP protocol. Storage servers may include their own locking semantics to arbitrate multiple device engine access to the same server without the need for a separate management server. Each device engine can access only a specific user's data package storage area even though the storage server 1415 may maintain a larger number of data packages across a large number of users. This allows for increased scaling when the storage server is implemented using file system techniques.

In one aspect, the storage server is implemented using standard FTP or HTTP connections for each operation. HTTP is composed of request response pairs. All requests are supposed to be posting commands. Parameters can be set in the form known as "application/X-WWW-form-URLENCODED". The encoding is specified as in RFC1866. Functions for the storage server include testing if the storage server can reach other users which will retrieve a simple text string, a "get" command which transfers the contents of a file as in a binary stream of bytes; a put command as a binary stream of data to the storage server, a directory listing command, a remove command, a rename command, an exist command, and the like.

Pull Synchronization

Figure 15 represents a "pull" synchronization process in accordance with the present invention. Both the pull synchronization illustrated in Figure 15 and the push synchronization illustrated in Figure 16 are done from the perspective of the device engine.

A pull synchronization as illustrated in Figure 15 is always performed

prior to a push synchronization. This allows the device engine to know whether synchronization of its own data is necessary.

Each device has its own triggering mechanism for initiating synchronization. Some devices, such as Windows clients and Palm® pilots are triggered manually when the user presses a "sync" button. Other devices, such as a cellular telephone, may be triggered automatically after another device completes a sync. Regular, time-based triggers are supported as well. A web-based application portal will sync when a user logs into the website security authorization mechanism, and may optionally sync on a log-out of the user or on the session time-out, but only if the user has changed data during the session.

For each sync, the triggering event specifies which application types are to sync for the device. This enables a triggering event to trigger only a sync for a particular application type. The management server can specify that no sync is needed for a particular type of application to minimize traffic to the storage server. Syncs may be triggered via an HTTP request to the server. This request holds information about which device to sync and the user log-in information is bounced to the management server for authorization and validation. Syncs may be triggered by sending an HTTP request to the server and passing the authentication information in the data portion of the request to the management server. Each device may include a servlet that is responsible for retrieving the request and ensuring its proper format before passing the synchronization request on to the server.

The device name and device class uniquely identify a particular device type that is being synchronized, and is contained in the management server. Each user has one or more device entries in the management server authorization records and each device name is unique for this user's space. For example, if a user has five devices with his or her own personal identification number, there will be five authorization records. There may be two Windows devices, two different Palm® devices and a web service portal,

each having their own personal identification number.

As shown in Figure 15, the pull synchronization process starts at an idle state 1405 when the triggering event, described above, triggers a synchronization request. The synchronization request is confirmed at 1410 and if the request is verified, a connection is made to the storage server at step 1415. Once a connection is established, the connection to the management server is made at step 1420 to authenticate the user identification via the management server. If authentication is successful, the management server may initiate a management server lock on the storage server so that no conflicting device engines may couple to the same data at the same time. A failure at any of the steps 1410-1425 will return the system to its idle state 1405. Once the engine server lock is acquired, the storage server will be checked to determine whether a new version of the data exists on the storage server at step 1430. If no new version exists, the synchronization process ends.

If a new version of the data exists, the device engine will retrieve the difference information at step 1435 "to get Δ ."

Once a Δ is retrieved, conflicts are resolved at step 1450. The resolve conflicts step allows a user to resolve conflicts to multiple types of data which have been changed on both the server portion of the device and in the local data.

Once the conflicts have been resolved at step 1450, the Δ 's are applied at step 1455. The apply Δ step 1455 allows for filters and mappings to be accounted for on the local device engine side of the system. As shown at steps 1460, 1465, 1470, and 1475, the Δ may include updates at the item level 1460, application level 1465, device level 1470, or network level 1475. In each of the aforementioned steps, a loop back to the Δ retrieval step 1435 is provided. When no further Δ 's are available, the management server lock is released at step 1440.

The foregoing description of a pull synchronization is further described

in the following pseudo-code:

```

*SymbolicSyncEngine::Sync
5      Download Remote File System
      For each Symbolic app in the file system's list of symbolic apps
10      CFDESymbolicSyncEngine::SyncSymbolicApp
      Create a structured delta object -- CStructuredDelta
      delta(...)
15      Compare local and remote versions of deltas SODs),
      if not the same then
          while localVersion != remoteVersion
20              download remote version
                  // apply delta (change log)
                  delta.ApplyChangeLog
                  // See details below
25              increment local version
          end while
30      else
          nothing to do
      end if
35      if any local items (change logs) are unsent then
          delta->ApplyUnsentItems (Reads the changes
[JCL where applied?])
40      end if
          // Generate a new change log for the device:
          delta->delta.GenerateChangeLog(... strChangeLogFile
45      ...      // See details below
          FTP it back up to Storage Server
          Update the local version number
50      end // SymbolicSyncEngine::SyncSymbolicApp
end // CFDESymbolicSyncEngine::Sync
55      CStructuredDelta::ApplyChangeLog
          Set up m_pAppObj;           // IFAO pointer
          Set up m_pAOS;             // IAOS pointer
60      Other set up (statistics, time to complete, etc.)
          Read the change log and ...
          ApplyChangeListToAOS(flChanges)
          for each item in list
65              ApplyItemToAOS // (Does

```

```

m_pAOS...AddRecord/UpdateRecord/DeleteRecord)
    end for

    end // ApplyChangeListToAOS
5      If not doing a full sync, also add changes from this file to
    apply these to m_FlChanges

10   end //CStructuredDelta::ApplyChangeLog

CStructuredDelta::GenerateChangeLog

15   Set up m_pAppObj;           // IFAO pointer
    Set up m_pAOS;              // IAOS pointer

    Other set up (statistics, time to complete, etc.)

20   // Set up m_deviceChanges by call to:
    CStructuredDelta::CreateDeviceChangeList

        Create a CFlItem* pItem

25   // Iterate FAO modifications:
        for (m_pAppObj->GetFirstModified(pItem),
m_pAppObj->GetNextModified(pItem))

        cast pItem to --> CFlItemUniversal* pUnItem

30   // Do certain things based on whether the operation
    is an add, delete, or update.
        // Then in each case, call:

35   CStructuredDelta::GetMatchingItemFromAOS

        // First get by FlID
        m_pAOS->GetRecordByFlID

40   // See if we have an AppID, if so:
        m_pAOS->GetRecordByAppID

        // If we can build search key on it
        iterate m_pAOS->GetFirstMatchingRecord

45   / m_pAOS->GetNextMatchingRecord

        end //
50   CStructuredDelta::GetMatchingItemFromAOS

        end for

        end // CStructuredDelta::CreateDeviceChangeList

55   if m_deviceChanges is not empty

        // reconcile (compare) change lists while writing to AOS
        CStructuredDelta::ReconcileChangeLists
        For each item in m_deviceChanges...
            If we can find it in m_FlChanges
                Reconcile the device item and the fl
            item
                end if
                ApplyItemToAOS // (Does
65   m_pAOS...AddRecord/UpdateRecord/DeleteRecord)
            end for

```



```

end // CStructuredDelta::ReconcileChangeLists

5          // Create a new change log (F1 delta package)
          ApplyChangeListToFl(m_deviceChanges)
            m_deviceChange.Store
            // Fires off its own whole world, see
FlItemList.cpp
10          end // m_deviceChange.Store

          end // ApplyChangeListToFl(m_deviceChanges)

          report stats

15        end if

          // Switch (SyncMode)
          If SyncMode == full
20            ApplyAOSToDevice
              iterate
m_pAOS->GetFirstRecord...m_pAOS->GetNextRecord
              Add or update the corresponding FAO record.
              (Note. Never delete based on what's in AOS, apparently).
25            end // ApplyAOSToDeviad
          else
            ApplyChangeListToDevice(m_flChanges);

End // CStructuredDelta::GenerateChangeLog

```

Push Synchronization

Figure 16 shows a push synchronization in accordance with the system and method of the present invention. Beginning at idle state 1505, a synchronization event occurs and if confirmed at step 1510, Δ 's are checked at step 1515. Depending on which type of changes occurred, a network Δ 1520, device Δ 1525, location Δ 1530, or item Δ 1535 will be created.

Once the Δ 's for a given application have been created, the method of the present invention continues at step 1540, which enables a connection to a storage server. Upon connection to the storage server, a further connection to management server 1545 will occur to authenticate the user in the system. Failure at any of the aforementioned points will result in returning to idle state 1505. Upon authentication, a management server lock is enabled to ensure that multiple device engines do not connect to the same data at the same time.

Once a lock is acquired at step 1555, Δ 's are uploaded to the system.

As shown, this may include uploading an item Δ 1575, an application Δ 1570, uploading a device Δ 1565, or a network Δ 1560. Once Δ 's have been uploaded to the server, management lock server 1580 is released, and the connection to the storage server is terminated at step 1585.

5 It should be recognized that such a push synchronization need not occur directly to a server, but may occur directly to a second device engine in accordance with the depiction of the multiple embodiments of the invention in Figures 1-7.

10 Data Package Specification

Once information is provided into the universal data format, the device engine organizes the format into a data package. Each data package thus includes a description of changes to any and all information for particular application, and a collection of data packages describes changes across all
15 device engines including all different types of data. With encoding and compression, data packages can become very compact to minimize bandwidth and storage requirements across the system of the present invention.

In one particular aspect of the present invention, encoding of the data
20 packages may be provided in a streaming format to allow processing by the device engines with minimal storage and memory configuration at the device engine level.

The device engine can read the stream and determine which records
25 from which applications it needs to update the particular information present on the system on which it resides.

Data packages can be provided in a binary data format. This allows
30 data packages to encode changes to non-application data at a bite level. Hence, if a single bit on a system changes, the system of the present invention allows synchronization of that bit on another system. Changes are described as a sequence of bite-level change operations. One such

encoding is using a sequence of insert and copy operations. Insert and copy operations generally define a particular "insertion" of a number of bites from a source file, then how many bites of a changed source file must be inserted to a particular file, then how many bites to insert from a particular new file, with a differencing engine taking the bites in the stream and inserting them into the new file to create the new version of the file.

As will be readily understood by one of average skill in the art, this allows a user to, for example, change a binary file such as a word processing document or other type of attachment, and synchronize such an attachment at the binary level. Specifically, if one forwards an e-mail of a word document to a second individual, the second individual modifies it and wishes to return this document with modifications to the first individual, because the first individual has the original file on his system, if both systems are enabled in the system of the present invention, the second system need only send the changes or the difference information back to the first system in order for the first system to reconstruct the document on the second system using this change data to create the document as intended by the second user.

Multiple caching of both the generation and application of data packages can be utilized to deal with communication issues in accordance with the system of the present invention. It should be further recognized that data packages can be merged into larger meta-data packages. Such meta-data information, such as the organization of multiple device packages, may be encoded into a larger system package. Each system package is essentially an encoded sequence of data packages.

Figure 12 shows the general format of the data package and universal data format an object stream hierarchy used in accordance with the present invention. With reference to Figures 11 and 12, one will note that each item in a particular application data structure will have a particular classification, such as a file, folder, contact, e-mail, calendar, etc. as shown in Figure 13. The universal data structure contains a mapped item field for each type of

data possible from each application supported by the system. Hence a "master" list of every data field mapping possible will contain a large number of items. Each application object requires a subset of such fields. One exception is an application object used for a Web portal application which provides access to all information available on all devices, including other Web portals.

Particular examples of item fields 1260 which may be included for any given item 1250 are shown in Figure 13. These exemplary item objects may, for example, be from an allocation such as Microsoft Outlook. Outlook allows for note items 1310, e-mail items 1320, task items 1330, calendar items 1340, bookmark items 1350, file items 1360, channel items 1370, folder items 1380, and contact items 1390, all of which have fields such as those represented in Figure 13.

The data format also contains folder information 1240 which allows the classification of items and consequently their associated item fields into particular categories.

Application objects 1230 include information on the types of applications from which information in the stream is included. Device objects 1220 include information on the origin type of device which the information is originating from. Network objects 1210 include information on a user level to define that the information in the data stream is coming from a particular user.

As detailed above, each application object supports a folder store interface that permits management of collections of information on a folder level, and permits management of folder hierarchies of information. The application object also includes an item interface that permits management of individual information entries such as records or files or components of information entries such as fields within records. Each application object further supports an interface for detection of a vendor application.

A DataPack essentially contains a sequence of transactions describing

changes to information. This information can span two basic types: structured or application data, and unstructured or binary file data.

Transactions are encoded using an efficient streaming format with tags to represent the actual content objects. This technique permits the
5 continuous extension of the DataPack format as new content is supported.

The general architecture of the package provides for transactions, application data, file data, files, objects and identifiers to be carried in the data package. Generally, transactions, application data, file data, and files have previously been described.

The first portion of the data package will be the data package
10 identifier. Each transaction has a basic architecture of objects and operations. Each piece of content is referred to as an object and is uniquely represented with a Universally Unique Identifier (UUID). Objects typically are represented by a dynamically generated UUID, but more common objects are
15 represented by static UUIDs. The following static UUIDs are defined:

20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100
105
110
115
120
125
130
135
140
145
150
155
160
165
170
175
180
185
190
195
200
205
210
215
220
225
230
235
240
245
250
255
260
265
270
275
280
285
290
295
300
305
310
315
320
325
330
335
340
345
350
355
360
365
370
375
380
385
390
395
400
405
410
415
420
425
430
435
440
445
450
455
460
465
470
475
480
485
490
495
500
505
510
515
520
525
530
535
540
545
550
555
560
565
570
575
580
585
590
595
600
605
610
615
620
625
630
635
640
645
650
655
660
665
670
675
680
685
690
695
700
705
710
715
720
725
730
735
740
745
750
755
760
765
770
775
780
785
790
795
800
805
810
815
820
825
830
835
840
845
850
855
860
865
870
875
880
885
890
895
900
905
910
915
920
925
930
935
940
945
950
955
960
965
970
975
980
985
990
995
1000
1005
1010
1015
1020
1025
1030
1035
1040
1045
1050
1055
1060
1065
1070
1075
1080
1085
1090
1095
1100
1105
1110
1115
1120
1125
1130
1135
1140
1145
1150
1155
1160
1165
1170
1175
1180
1185
1190
1195
1200
1205
1210
1215
1220
1225
1230
1235
1240
1245
1250
1255
1260
1265
1270
1275
1280
1285
1290
1295
1300
1305
1310
1315
1320
1325
1330
1335
1340
1345
1350
1355
1360
1365
1370
1375
1380
1385
1390
1395
1400
1405
1410
1415
1420
1425
1430
1435
1440
1445
1450
1455
1460
1465
1470
1475
1480
1485
1490
1495
1500
1505
1510
1515
1520
1525
1530
1535
1540
1545
1550
1555
1560
1565
1570
1575
1580
1585
1590
1595
1600
1605
1610
1615
1620
1625
1630
1635
1640
1645
1650
1655
1660
1665
1670
1675
1680
1685
1690
1695
1700
1705
1710
1715
1720
1725
1730
1735
1740
1745
1750
1755
1760
1765
1770
1775
1780
1785
1790
1795
1800
1805
1810
1815
1820
1825
1830
1835
1840
1845
1850
1855
1860
1865
1870
1875
1880
1885
1890
1895
1900
1905
1910
1915
1920
1925
1930
1935
1940
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
2005
2010
2015
2020
2025
2030
2035
2040
2045
2050
2055
2060
2065
2070
2075
2080
2085
2090
2095
2100
2105
2110
2115
2120
2125
2130
2135
2140
2145
2150
2155
2160
2165
2170
2175
2180
2185
2190
2195
2200
2205
2210
2215
2220
2225
2230
2235
2240
2245
2250
2255
2260
2265
2270
2275
2280
2285
2290
2295
2300
2305
2310
2315
2320
2325
2330
2335
2340
2345
2350
2355
2360
2365
2370
2375
2380
2385
2390
2395
2400
2405
2410
2415
2420
2425
2430
2435
2440
2445
2450
2455
2460
2465
2470
2475
2480
2485
2490
2495
2500
2505
2510
2515
2520
2525
2530
2535
2540
2545
2550
2555
2560
2565
2570
2575
2580
2585
2590
2595
2600
2605
2610
2615
2620
2625
2630
2635
2640
2645
2650
2655
2660
2665
2670
2675
2680
2685
2690
2695
2700
2705
2710
2715
2720
2725
2730
2735
2740
2745
2750
2755
2760
2765
2770
2775
2780
2785
2790
2795
2800
2805
2810
2815
2820
2825
2830
2835
2840
2845
2850
2855
2860
2865
2870
2875
2880
2885
2890
2895
2900
2905
2910
2915
2920
2925
2930
2935
2940
2945
2950
2955
2960
2965
2970
2975
2980
2985
2990
2995
3000
3005
3010
3015
3020
3025
3030
3035
3040
3045
3050
3055
3060
3065
3070
3075
3080
3085
3090
3095
3100
3105
3110
3115
3120
3125
3130
3135
3140
3145
3150
3155
3160
3165
3170
3175
3180
3185
3190
3195
3200
3205
3210
3215
3220
3225
3230
3235
3240
3245
3250
3255
3260
3265
3270
3275
3280
3285
3290
3295
3300
3305
3310
3315
3320
3325
3330
3335
3340
3345
3350
3355
3360
3365
3370
3375
3380
3385
3390
3395
3400
3405
3410
3415
3420
3425
3430
3435
3440
3445
3450
3455
3460
3465
3470
3475
3480
3485
3490
3495
3500
3505
3510
3515
3520
3525
3530
3535
3540
3545
3550
3555
3560
3565
3570
3575
3580
3585
3590
3595
3600
3605
3610
3615
3620
3625
3630
3635
3640
3645
3650
3655
3660
3665
3670
3675
3680
3685
3690
3695
3700
3705
3710
3715
3720
3725
3730
3735
3740
3745
3750
3755
3760
3765
3770
3775
3780
3785
3790
3795
3800
3805
3810
3815
3820
3825
3830
3835
3840
3845
3850
3855
3860
3865
3870
3875
3880
3885
3890
3895
3900
3905
3910
3915
3920
3925
3930
3935
3940
3945
3950
3955
3960
3965
3970
3975
3980
3985
3990
3995
4000
4005
4010
4015
4020
4025
4030
4035
4040
4045
4050
4055
4060
4065
4070
4075
4080
4085
4090
4095
4100
4105
4110
4115
4120
4125
4130
4135
4140
4145
4150
4155
4160
4165
4170
4175
4180
4185
4190
4195
4200
4205
4210
4215
4220
4225
4230
4235
4240
4245
4250
4255
4260
4265
4270
4275
4280
4285
4290
4295
4300
4305
4310
4315
4320
4325
4330
4335
4340
4345
4350
4355
4360
4365
4370
4375
4380
4385
4390
4395
4400
4405
4410
4415
4420
4425
4430
4435
4440
4445
4450
4455
4460
4465
4470
4475
4480
4485
4490
4495
4500
4505
4510
4515
4520
4525
4530
4535
4540
4545
4550
4555
4560
4565
4570
4575
4580
4585
4590
4595
4600
4605
4610
4615
4620
4625
4630
4635
4640
4645
4650
4655
4660
4665
4670
4675
4680
4685
4690
4695
4700
4705
4710
4715
4720
4725
4730
4735
4740
4745
4750
4755
4760
4765
4770
4775
4780
4785
4790
4795
4800
4805
4810
4815
4820
4825
4830
4835
4840
4845
4850
4855
4860
4865
4870
4875
4880
4885
4890
4895
4900
4905
4910
4915
4920
4925
4930
4935
4940
4945
4950
4955
4960
4965
4970
4975
4980
4985
4990
4995
5000
5005
5010
5015
5020
5025
5030
5035
5040
5045
5050
5055
5060
5065
5070
5075
5080
5085
5090
5095
5100
5105
5110
5115
5120
5125
5130
5135
5140
5145
5150
5155
5160
5165
5170
5175
5180
5185
5190
5195
5200
5205
5210
5215
5220
5225
5230
5235
5240
5245
5250
5255
5260
5265
5270
5275
5280
5285
5290
5295
5300
5305
5310
5315
5320
5325
5330
5335
5340
5345
5350
5355
5360
5365
5370
5375
5380
5385
5390
5395
5400
5405
5410
5415
5420
5425
5430
5435
5440
5445
5450
5455
5460
5465
5470
5475
5480
5485
5490
5495
5500
5505
5510
5515
5520
5525
5530
5535
5540
5545
5550
5555
5560
5565
5570
5575
5580
5585
5590
5595
5600
5605
5610
5615
5620
5625
5630
5635
5640
5645
5650
5655
5660
5665
5670
5675
5680
5685
5690
5695
5700
5705
5710
5715
5720
5725
5730
5735
5740
5745
5750
5755
5760
5765
5770
5775
5780
5785
5790
5795
5800
5805
5810
5815
5820
5825
5830
5835
5840
5845
5850
5855
5860
5865
5870
5875
5880
5885
5890
5895
5900
5905
5910
5915
5920
5925
5930
5935
5940
5945
5950
5955
5960
5965
5970
5975
5980
5985
5990
5995
6000
6005
6010
6015
6020
6025
6030
6035
6040
6045
6050
6055
6060
6065
6070
6075
6080
6085
6090
6095
6100
6105
6110
6115
6120
6125
6130
6135
6140
6145
6150
6155
6160
6165
6170
6175
6180
6185
6190
6195
6200
6205
6210
6215
6220
6225
6230
6235
6240
6245
6250
6255
6260
6265
6270
6275
6280
6285
6290
6295
6300
6305
6310
6315
6320
6325
6330
6335
6340
6345
6350
6355
6360
6365
6370
6375
6380
6385
6390
6395
6400
6405
6410
6415
6420
6425
6430
6435
6440
6445
6450
6455
6460
6465
6470
6475
6480
6485
6490
6495
6500
6505
6510
6515
6520
6525
6530
6535
6540
6545
6550
6555
6560
6565
6570
6575
6580
6585
6590
6595
6600
6605
6610
6615
6620
6625
6630
6635
6640
6645
6650
6655
6660
6665
6670
6675
6680
6685
6690
6695
6700
6705
6710
6715
6720
6725
6730
6735
6740
6745
6750
6755
6760
6765
6770
6775
6780
6785
6790
6795
6800
6805
6810
6815
6820
6825
6830
6835
6840
6845
6850
6855
6860
6865
6870
6875
6880
6885
6890
6895
6900
6905
6910
6915
6920
6925
6930
6935
6940
6945
6950
6955
6960
6965
6970
6975
6980
6985
6990
6995
7000
7005
7010
7015
7020
7025
7030
7035
7040
7045
7050
7055
7060
7065
7070
7075
7080
7085
7090
7095
7100
7105
7110
7115
7120
7125
7130
7135
7140
7145
7150
7155
7160
7165
7170
7175
7180
7185
7190
7195
7200
7205
7210
7215
7220
7225
7230
7235
7240
7245
7250
7255
7260
7265
7270
7275
7280
7285
7290
7295
7300
7305
7310
7315
7320
7325
7330
7335
7340
7345
7350
7355
7360
7365
7370
7375
7380
7385
7390
7395
7400
7405
7410
7415
7420
7425
7430
7435
7440
7445
7450
7455
7460
7465
7470
7475
7480
7485
7490
7495
7500
7505
7510
7515
7520
7525
7530
7535
7540
7545
7550
7555
7560
7565
7570
7575
7580
7585
7590
7595
7600
7605
7610
7615
7620
7625
7630
7635
7640
7645
7650
7655
7660
7665
7670
7675
7680
7685
7690
7695
7700
7705
7710
7715
7720
7725
7730
7735
7740
7745
7750
7755
7760
7765
7770
7775
7780
7785
7790
7795
7800
7805
7810
7815
7820
7825
7830
7835
7840
7845
7850
7855
7860
7865
7870
7875
7880
7885
7890
7895
7900
7905
7910
7915
7920
7925
7930
7935
7940
7945
7950
7955
7960
7965
7970
7975
7980
7985
7990
7995
8000
8005
8010
8015
8020
8025
8030
8035
8040
8045
8050
8055
8060
8065
8070
8075
8080
8085
8090
8095
8100
8105
8110
8115
8120
8125
8130
8135
8140
8145
8150
8155
8160
8165
8170
8175
8180
8185
8190
8195
8200
8205
8210
8215
8220
8225
8230
8235
8240
8245
8250
8255
8260
8265
8270
8275
8280
8285
8290
8295
8300
8305
8310
8315
8320
8325
8330
8335
8340
8345
8350
8355
8360
8365
8370
8375
8380
8385
8390
8395
8400
8405
8410
8415
8420
8425
8430
8435
8440
8445
8450
8455
8460
8465
8470
8475
8480
8485
8490
8495
8500
8505
8510
8515
8520
8525
8530
8535
8540
8545
8550
8555
8560
8565
8570
8575
8580
8585
8590
8595
8600
8605
8610
8615
8620
8625
8630
8635
8640
8645
8650
8655
8660
8665
8670
8675
8680
8685
8690
8695
8700
8705
8710
8715
8720
8725
8730
8735
8740
8745
8750
8755
8760
8765
8770
8775
8780
8785
8790
8795
8800
8805
8810
8815
8820
8825
8830
8835
8840
8845
8850
8855
8860
8865
8870
8875
8880
8885
8890
8895
8900
8905
8910
8915
8920
8925
8930
8935
8940
8945
8950
8955
8960
8965
8970
8975
8980
8985
8990
8995
9000
9005
9010
9015
9020
9025
9030
9035
9040
9045
9050
9055
9060
9065
9070
9075
9080
9085
9090
9095
9100
9105
9110
9115
9120
9125
9130
9135
9140
9145
9150
9155
9160
9165
9170
9175
9180
9185
9190
9195
9200
9205
9210
9215
9220
9225
9230
9235
9240
9245
9250
9255
9260
9265
9270
9275
9280
9285
9290
9295
9300
9305
9310
9315
9320
9325
9330
9335
9340
9345
9350
9355
9360
9365
9370
9375
9380
9385
9390
9395
9400
9405
9410
9415
9420
9425
9430
9435
9440
9445
9450
9455
9460
9465
9470
9475
9480
9485
9490
9495
9500
9505
9510
9515
9520
9525
9530
9535
9540
9545
9550
9555
9560
9565
9570
9575
9580
9585
9590
9595
9600
9605
9610
9615
9620
9625
9630
9635
9640
9645
9650
9655
9660
9665
9670
9675
9680
9685
9690
9695
9700
9705
9710
9715
9720
9725
9730
9735
9740
9745
9750
9755
9760
9765
9770
9775
9780
9785
9790
9795
9800
9805
9810
9815
9820
9825
9830
9835
9840
9845
9850
9855
9860
9865
9870
9875
9880
9885
9890
9895
9900
9905
9910
9915
9920
9925
9930
9935
9940
9945
9950
9955
9960
9965
9970
9975
9980
9985
9990
9995
10000
10005
10010
10015
10020
10025
10030
10035
10040
10045
10050
10055
10060
10065
10070
10075
10080
10085
10090
10095
10100
10105
10110
10115
10120
10125
10130
10135
10140
10145
10150
10155
10160
10165

specific files based on the following rules:

- Transactions related to account information are grouped into a DataPack file.
- Transactions related to a specific data class are grouped into a DataPack file.
- Transactions referring to binary data are grouped into separate DataPack files for each file object.

A DataPack file is identified using specific rules based on the file name. The file name is of the form "UUID.VER" where UUID is the identifier for the specific object and VER is the transaction version number. The version number is of the form "D0001" with additional digits used for large version numbers. The "D000" value may be reserved for the base version for the object.

The UUID for the user account is generated by the Management Server (MS). The MS also maintains a current table of UUID values and version numbers that provides the root structure for understanding the DataPack files within a user account. The MS also provides necessary locking semantics needed to maintain consistency when multiple device engines attempt to synchronize.

All DataPacks are prefixed with a standardized header that provides basic content information regarding the DataPack. Compression and encryption headers follow the DataPack header if needed.

The data package header information will include version signature, applied versioning information, content type, Δ engine type, compression type, encryption type, applied size, encrypted size, compressed size, raw data size, and other data useful for the device engine in decrypting the data stream to provide the data into a format usable for the application.

The header may optimally have the format:

5

10

15

20

Type	Bytes
Version	4
Signature	4
AppliedVersion	8
ContentType	4
DeltaType	4
CompressionType	4
EncryptionType	4
AppliedSize	4
EncryptedSize	4
CompressedSize	4
RawSize	4
Reserved	TBD

The following ContentType values are permissible:

Field	Comment
DP_CONTENT_RAW	Raw
DP_CONTENT_COMPRESSED	Compressed
DP_CONTENT_ENCRYPTED	Encrypted

25

30

The DeltaType encodes the type of binary file differencing used. The following DeltaType values are permissible using DataPackageDeltaType:

Field	Comment
PackageDeltaTypeUninitialized	Uninitialized
PackageDeltaTypeRawData	Raw binary data
PackageDeltaTypeDeltaXDelta	Xdelta binary difference
PackageDeltaTypeDeltaBDiff	Bdiff binary difference

The compression type specifies whether the DataPack has been compressed. A DataPack compression header follows the DataPack header if a compression type is specified. The following CompressionType values are permissible using DataPackageCompressionType:

5

Field	Comment
PackageCompressionTypeUninitialized	Uninitialized
PackageCompressionTypeNone	None
PackageCompressionTypePK	PKZip format
PackageCompressionTypeLZS	LZS format

10

The encryption type specifies whether the DataPack has been encrypted. A DataPack encryption header follows the DataPack header if an encryption type is specified. The following EncryptionType values are permissible using DataPackageEncryptionType:

15

Field	Comment
PackageEncryptionTypeUninitialized	Uninitialized
PackageEncryptionTypeNone	None
PackageEncryptionTypeXORTest	XOR masked data
PackageEncryptionTypeBlowFish	Blowfish
PackageEncryptionTypeTwoFish	Twofish

20

All DataPack compression headers are encoded using the following format:

25

Field	Size (bytes)	Comment
Size	4	Size of data including this header
Version	4	Version (1)
Signature	4	Signature (4271)
HeaderType	4	Header type (HeaderTypeCompression)

30

Reserved	12	Reserved
DecompressedSize	4	Decompressed size
Reserved	50	Reserved
Reserved	12	Reserved

5

The following HeaderType values are permissible using DataPackageHeaderType:

Field	Comment
HeaderTypeUninitialized	Uninitialized
HeaderTypeEncryption	Encryption header
HeaderTypeCompression	Compression header
HeaderTypeRaw	Raw header

10

15

All DataPack encryption headers are encoded using the following format:

Field	Size (bytes)	Comment
Size	4	Size of data including this header
Version	4	Version (6)
Signature	4	Signature (4270)
HeaderType	4	Header type (HeaderTypeEncryption)
Reserved	12	Reserved
DecryptedSize	4	Decrypted size
InitValue	16	TBD

20

25

KeyLength	4	TBD
ClearTextKeyBits	4	TBD
Salt	4	TBD
PadBytes	4	TBD
HMAC	20	TBD
Reserved	12	Reserved

The data package transaction format may take a number of forms. One example is the following:

DataPack transaction format - header - info - objects and operations
Diagram

transaction ::= fileData | Header + InfoList + TransactionList

fileData ::= raw binary file data | binary difference data

Header ::= ID + DataPackID + DataPackVersion

ID ::= FUSE

DataPackID ::= CLOG

InfoList ::= FieldList

TransactionList ::= Operation + [ItemInfo + [FieldList]]

Operation ::= see table below

ItemInfo ::= ItemType + ItemFlags + EntryID + ParentEntryID

ItemType ::= same as enumItemType

ItemFlags ::= same as enumItemFlags

EntryID ::= UUID

ParentEntryID ::= UUID

UUID ::= 128-bit UUID as defined by standard

FieldList ::= [FieldTag + FieldData] + ListEnd

FieldTag ::= same as enumFieldTag

FieldData ::= FieldDataType + [FieldDataLen + [FieldDataData]]

FieldDataType ::= see below

FieldDataLen ::= length as required by FieldDataType

FieldDataData ::= data as required by FieldDataType

ListEnd ::= DWORD(0)

The following Operation values are permissible using the Operation class:

Field	Comment
clNop	None
clAdd	Add
clDelete	Delete

5

clChange	Change
clMove	Move
clRename	Rename
clForceChange	Force change without conflict

The following FieldDataType values are permissible using clDataType:

10

15

20

25

30

Field	Comment
clInvalidType	TBD
clString	Unicode String bytes with a 32-bit length prefix
clString8	Unicode String bytes with an 8-bit length prefix
clString16	Unicode String bytes with a 16-bit length prefix
clEmpty String	TBD
clBlob	32-bit length followed by a byte stream
clBlob8	8-bit length followed by a byte stream
clBlob16	16-bit length followed by a byte stream
clEmptyBlob	TBD
clByte	8-bit value
clShort	16-bit value
clDword	32-bit value
clQword	64-bit value
clDate	DATE type (double)
clDouble	8 byte real
clFloat	4 byte real
clUuid	16 byte uuid
clZero	Zero value
clOne	One value
clUnspecified	Unspecified value
clDefault	Default value
clCollection	Collection with 32-bit length

clCollection8	Collection with 8-bit length
clCollection 16	Collection with 16-bit length
clEmptyCollection	Collection with no length

5 Data package objects are organized into a hierarchy as follows:

```
Account ::= DeviceList + DataClassList
DeviceList ::= {Device}
DataClassList ::= {DataClass} + ProviderList
10 ProviderList ::= {Provider} + DataStoreList
DataStoreList ::= {Folder} + ItemList
ItemList ::= {Item} + FieldList
FieldList ::= {Field}
```

15 An account is the root structure, which identifies information about the user's account. It may have exemplary field tags (eFieldTag_[NAME]) such as Name, Password, UserName and Version. The FieldTag ItemType value is specified as ItemType_PIN using enumItemType.

20 A device is a system identified as part of an account. Examples include PCs, handhelds, Web sites, and so on. It may have tags (eFieldTag_[Name]) such as: "name" and "type" and item type values (eDevice_[Name]) such as Portal, Palm, Windows, 25 CellPhone.

A data class is a grouping of similar information types. Many data classes may be represented for a particular account. The data class may contain field tags (eFieldTag_[Name]) such as: Name; ItemType; 30 SubType; IsManaged; Provider; Filter and Version.

The following ItemType values are permissible using enumDataClass (eDataClass_[Name]):

<u>Tag</u>	<u>Description</u>
UNKNOWN	Unknown

	CONTACT	Contact/address book
	EMAIL	Electronic mail
	CALENDAR	Calendar
5	TASK	Task/to do
	NOTE	Note/memo
	JOURNAL	Journal
	BROWSER	Web browser favorites, cookies, etc.
	FILESET	Collection of files
	PIN	Account information
10	DEVICE	Device information
	FILEBODY	Contents of file

15 A Provider is the application that maintains specific information within a data class. There can be more than one provider for a particular data class. Field tags include: Name, AppObjID, Password, Username and Version. Examples of provider tags permissible for the provider (eProvider[Name]) include: Portal, Palm®, MicrosoftOutlook®, Lotus Organizer, Microsoft Internet Explorer, Microsoft Windows, and so on.

20 Data stores are the containers for storing information within a provider. There can be more than one data store for a particular provider. Folders represent structural organization of information within a data store. Data stores are not required to support folders. Tags (eFieldTag_[Name]) supported for each data store include: Name, ItemType, IsManaged and OriginalPath. Item types permissible for the data store include: unknown; Folder; MAPI; Database and Store_File.

30 Folders represent structural organization of information within a data store. Data stores are not required to support folders. A folder is represented by a UUID and may contain any of the following field tags (eFieldTag_[Name]): Name; ItemType; IsManaged; FileAttributes; CreationDate; ModificationDate; AccessDate; SpecialFolderType.

The eFieldTag_ItemType value is specified as eltemType_FOLDER using enumItemType.

Items are individual informational components consisting of the actual user data. They may contain field tags such as: Name, ItemType, IsManaged, and Version.

File items typically have the following additional field tags (eFieldTag_[Name]):

FileAttributes
CreationDate
ModificationDate
AccessDate
FileSize
FileBody
DeltaSize
Hash

Item types may take the format (eltemType_[Name]) and may include: extended; folder; attachment; contact; distlist; email; calendar; task; call; note; post; journal; form; script; rule; favorites; subscription; common_favorites; desktop; common_desktop; startmenu; common_startmenu; channels; cookies; programs; common_programs; startup; common_startup; sendto; recent; internet_cache; history; mapped_drives; printers; docs; doctemplates; fonts; window_settings; app_data_folder; app_settings; fileset; pin; device; data_store; file; provider; and data_class; internal.

A field is based on one of a set of base type definitions. All field tag information is encoded using the following format:

Field	Size (bits)	Comment
FieldTag	16	Unique tag number
FieldType	6	Field base type
FieldSubType	10	Field sub-type

A number of Field types are possible, including: unknown; long; dword; date; string; binary; float; double; collection; uniqueid; qword; uuid; file; invalid. LONG is a four byte value encoded in big-endian format. FieldType DWORD is a four byte value encoded in big-endian format. FieldType String is a sequence of Unicode characters followed by a single NULL byte. Interfaces are provided with an MBCS value. FieldType Binary is a sequence of bytes. FieldType UniqueID is a sequence of bytes as defined by the Universally Unique Identifier (UUID) standard. AO interfaces are provided with a Locally Unique Identifier (LUID) value. FieldType QWORD is an eight byte value encoded in big-endian format. FieldType File is a UUID that references a separate DataPack containing the file body data. AO interfaces are provided with a sequence of Unicode characters followed by a single NULL byte that describes the full path name for the file.

Any number of filed sub types are possible. Each of the sub-types includes all of the possible data types from all of the supported user applications. As should be well understood, the possibilities in the number of sub-types is quite large, and dynamic as each new application supported by the system of the present invention is added. Examples of sub-types include:

<u>SubField Description</u>	<u>Description</u>
Base	No sub-type specified
EmailAddress	Email address
EmailAddressList	Email address list
SearchKey	Search key
CategoryList	Category list
StringList	String list
DistributionList	Distribution list
Gender	Gender (enumGender)
TimeZone	Time zone (enumTimeZone)
Boolean	Boolean (TBD)

	NonZeroBool	Boolean with non-zero value (enumNonZeroBool)
	Priority	Priority
5	Sensitivity	Sensitivity (enumSensitivity)
	Importance	Importance (enumImportance)
	SelectedMailingAddr	Selected mailing address (enumSelectedMailingAddr)
	TaskStatus	Task status (enumTaskStatus)
	FlagStatus	Flag status (enumFlagStatus)
10	RecurrenceType	Recurrence type (enumRecurrenceType)
	DayOfWeek	Day of week (enumDayOfWeek)
	DayOfMonth	Day of month (1 through 31)
	InstanceOfMonth	Instance of month (enumInstanceOfMonth)
15	MonthOfYear	Month of year (enumMonthOfYear)
	BusyStatus	Busy status (enumBusyStatus)
	AttachmentType	Attachment type (enumAttachmentType)
	MailBodyType	Mail body type (enumMailBodyType)
	RGB	RGB color value
20	ManagedState	Managed state (enumManagedState)
	FaoId	FAO ID for provider
	SpecialFolderType	Special folder type (enumSpecialFolderType)
	ResponseState	Response state (TBD)
25	ResponseStatus	Response status (TBD)
	JournalStatus	Journal status
	PageStyle	Page style
	PageNumberMethod	Page number method
	DelegationState	Delegation state
30	MeetingStatus	Meeting status
	MeetingInvitation	Meeting invitation
	CalendarType	Calendar type
	DateOnly	Date only
	TimeOnly	Time only
35	PhoneNumber	Phone number
	URL	URL
	FilePath	File path
	PopMessageID	POP message ID
	MIMEType	MIME type
40	INVALID	All values must be below this

The aforementioned invention provides a user-centric model of

communication to deliver personal information via network services. This model accommodates devices that are disconnected from the network, such as the Internet, at various times. Personal information can continue to exist locally rather than imposing a server-centric model on existing information.

In accordance with the foregoing, a store and forward information broadcast is utilized. Changes to existing information are replicated to an Internet storage server and changes are then retrieved by other devices on the network at device-specific times. In this manner, direct client communication is accomplished without requiring one-to-one communication. While one communication is supported by the system of the present invention, it need not be required.

Although the present invention has been presented in the form of an Internet store and forward broadcast for the purposes of synchronizing personal information amongst various types of devices, it will be readily recognized that synchronization need not be accomplished as the only application for the aforementioned system. In particular, the system can be utilized to efficiently broadcast changes to information in so-called "push" type information applications where only portions of the data need to be changed on a client application. For example, in a system where information such as changes in a stock price need to be broadcast to a plurality of users, a client application implementing the aforementioned technology can be updated by only changing specific portions of the data in the client application relative to that particular stock price. This can be done using a smaller bandwidth than has previously been determined with other devices.

The many objects and advantages of the present invention will be readily apparent to one of average skill in the art. All such objects and advantages are intended to be within the scope of the invention as defined

Attorney Docket No.: FUSN1-01002US0
lev/fusn1/1002.001.wpd

CLAIMS

What is claimed is:

1 1. An system for synchronizing data between a first system and
2 a second system, comprising:

3 a first sync engine on the first system interfacing with data on the
4 first system to provide difference information;

5 a data store coupled to network and in communication with the first
6 and second systems; and

7 a second sync engine on the second system coupled to receive the
8 difference information from the data store via the network, and interfacing
9 with data on the second system to update said data on the second system
10 with said difference information.

1 2. The apparatus of claim 1 wherein the first system and second
2 system are coupled to the server via a private network.

1 3. The apparatus of claim 1 wherein the first system and second
2 system are coupled to the server via an Internet connection.

1 4. The apparatus of claim 1 wherein the difference information
2 is transmitted to the data store by the first sync engine and received from
3 the data store from the second sync engine.

1 5. The apparatus of claim 4 wherein the difference information
2 is transmitted to the data store at a first point in time, and received from the
3 data store at a second, subsequent point in time.

1 6. The apparatus of claim 1 wherein said second sync engine
2 interfaces with said data on the second system to provide second
3 difference information to the data store.

1 7. The apparatus of claim 6 wherein the first sync engine
2 couples to the data store to retrieve the second difference information and
3 interfaces with the data on the first system to update said data on the first
4 system with said second difference information.

1 8. The apparatus of claim 1 further including a management
2 server coupled to the network and in communication with the first sync
3 engine, the second sync engine and the data store.

1 9. The apparatus of claim 8 wherein said management server
2 authorizes access of difference information on the data store by the first
3 and second sync engines.

1 10. The apparatus of claim 8 wherein said management server
2 locks access to difference information on the data store during
3 communication with the first and the second sync engines.

1 11. The apparatus of claim 1 further including a first device,
2 coupled to the first system via the network, providing said data to the first
3 system.

1 12. The apparatus of claim 11 wherein the first system is a sync

2 server.

1 13. The apparatus of claim 11 wherein said data comprises
2 changes to a previous state of the data, and said difference information
3 comprises said changes in an encoded, universal format.

1 14. The apparatus of claim 1 wherein said sync engine
2 comprises:

3 a data interface;
4 a copy of a previous state of said data; and
5 a difference transaction generator.

1 15. The apparatus of claim 1 wherein said data on said first
2 system comprises application data having a plurality of application specific
3 formats, and said difference information is provided for each of said
4 formats in a universal format to said data store.

1 16. The apparatus of claim 1 further including:
2 a plurality of sync engines on a respective plurality of systems, each
3 of said plurality of engines being coupled to receive difference information
4 from each of said first, second and plurality of sync engines from the data
5 store via the network, and each said engine interfacing with data on the
6 system on which it resides to update said data on said system on which it
7 resides with said difference information, and interface with data on said
8 system on which it resides to provide difference data information from the
9 system on which it resides to the data store.

1 17. A system, comprising:
2 a first device including at least a first data file and first differencing
3 code having an input and an output coupled to a network to receive first
4 device data change transactions, based on said at least one data file, from
5 and provide change transactions to, said network;
6 a data store coupled to the network having at least one data
7 structure coupled to store change transactions; and
8 a second system including at least a second data file and second
9 differencing code having an input and an output coupled to the network to
10 receive said first device data change transactions, and provide second
11 change transactions based on said at least second data file to said data
12 store.

1 18. The apparatus of claim 17 wherein the first device and
2 second device are coupled to the data store via an Internet connection.

1 19. The apparatus of claim 17 wherein the first change
2 transactions are transmitted to the data store by the first device at a first
3 point in time and received from the data store by the second device at a
4 second, subsequent point in time.

1 20. The apparatus of claim 17 wherein the first differencing code
2 receives second change transactions from the data store to and interfaces
3 with at least the first data file on to update said data with said second
4 change transactions.

1 21. The apparatus of claim 17 further including a management

2 server coupled to the network and in communication with the first sync
3 engine, the second sync engine and the data store.

1 22. The apparatus of claim 17 wherein said management server
2 authorizes access of difference information on the data store by the first
3 and second differencing code.

1 23. The apparatus of claim 17 wherein the first device is a sync
2 server.

1 24. The apparatus of claim 17 wherein said differencing code
2 comprises:
3 an application object;
4 an application object store; and
5 a delta engine.

1 25. A method for synchronizing at least a first and a second
2 resident on a first and a second systems, respectively, coupled to the
3 Internet, respectively, comprising:
4 determining difference data resulting from changes to the first file
5 on the first system;
6 transmitting the difference data to a server via the Internet;
7 querying the server from a second system to determine whether
8 difference data exists for files on the second system;
9 retrieving the difference data to the second system; and
10 updating the second file on the second system with the difference

11 data.

1 26. The method of claim 25 wherein said step of determining
2 comprises:

3 comparing a current instance of the first file to a stored instance of
4 the first file; and

5 generating said difference data.

1 27. The method of claim 25 wherein said step of querying
2 comprises:

3 coupling to a management server which provides information on a
4 state of said difference data for files on the first system and the second
5 system.

1 28. The method of claim 25 wherein said step of retrieving
2 comprises:

3 coupling to the management server;

4 receiving authorization from the management server to retrieve the
5 difference data;

6 transmitting a request to the server for the difference data; and

7 receiving the difference data in response to the request.

1 29. The method of claim 28 wherein the management server
2 locks the server from receiving further requests for the difference data
3 during said retrieving step.

1 30. The method of claim 25 wherein said step of updating
2 comprises:

3 applying the difference data to a stored instance of the second file
4 on the second system.

1 31. An Internet synchronization system, comprising:

2 a storage server having an Internet connection;

3 a first device coupled to the Internet and including a device sync
4 engine; and

5 a second device coupled to the Internet and including a second
6 device sync engine.

1 32. The Internet synchronization system of claim 31 further
2 including:

3 a management server.

1 33. The Internet synchronization system of claim 32 wherein the
2 storage server and the management server are provided in a data center.

1 34. The Internet synchronization system of claim 31 wherein
2 communications between the first device, the second device and the
3 storage server are encoded and compressed.

1 35. The Internet synchronization system of claim 31 wherein data
2 transfer between the first device, the second device and the storage server
3 comprises difference transactions.

1 36. The Internet synchronization system of claim 31 wherein
2 each device includes applications having data in an application specific
3 format, and wherein communication between the first device, the second
4 device and the storage server include changes to said data in an
5 application independent format.

1 37. The Internet synchronization system of claim 31 wherein
2 each device sync engine comprises:

3 an application object;
4 an application object store; and
5 a delta engine.

ABSTRACT

A system and method for synchronizing devices which can couple to the Internet, or any network. In one aspect a system for synchronizing data between a first system and a second system is provided. The system
5 includes a first sync engine on the first system interfacing with data on the first system to provide difference information. A data store is coupled to network and in communication with the first and second systems. A second sync engine is provided on the second system coupled to receive the difference information from the data store via the network, and
10 interfacing with data on the second system to update said data on the second system with said difference information.

Difference information is transmitted to the data store by the first sync engine and received from the data store from the second sync engine. The system may include a management server coupled to the network and in communication with the first sync engine, the second sync engine and the data store. The system may include a plurality of sync engines on a respective plurality of systems, each of said plurality of engines being coupled to receive difference information from each of said
15 first, second and plurality of sync engines from the data store via the network. Each said engine interfaces with data on the system on which it resides to update said data on said system on which it resides with said difference information, and interfaces with data on said system on which it resides to provide difference data information from the system on which it resides to the data store.

In a further embodiment, the invention comprises a method for
25 synchronizing at least a first file and a second file resident on a first and a second systems, respectively, coupled to the Internet, respectively. The method includes the steps of: determining difference data resulting from changes to the first file on the first system; transmitting the difference data

to a server via the Internet; querying the server from a second system to determine whether difference data exists for files on the second system; retrieving the difference data to the second system; and updating the second file on the second system with the difference data.

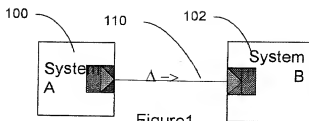


Figure 1

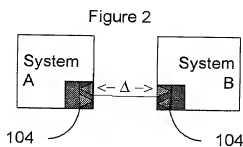


Figure 2

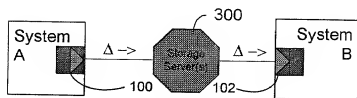


Figure 3

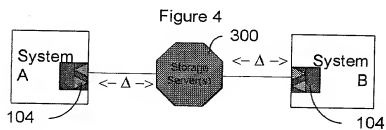


Figure 4

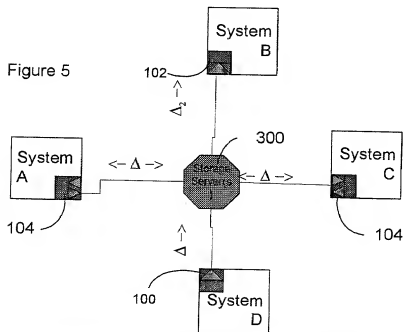


Figure 5

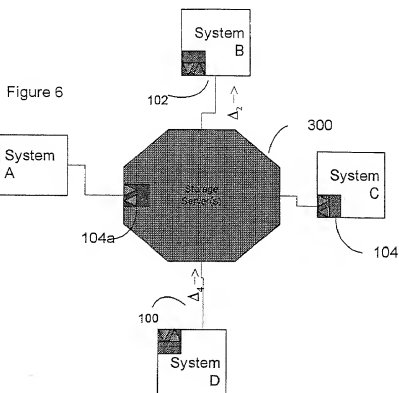


Figure 7

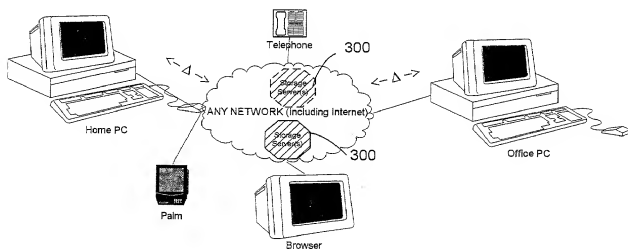


Figure 8

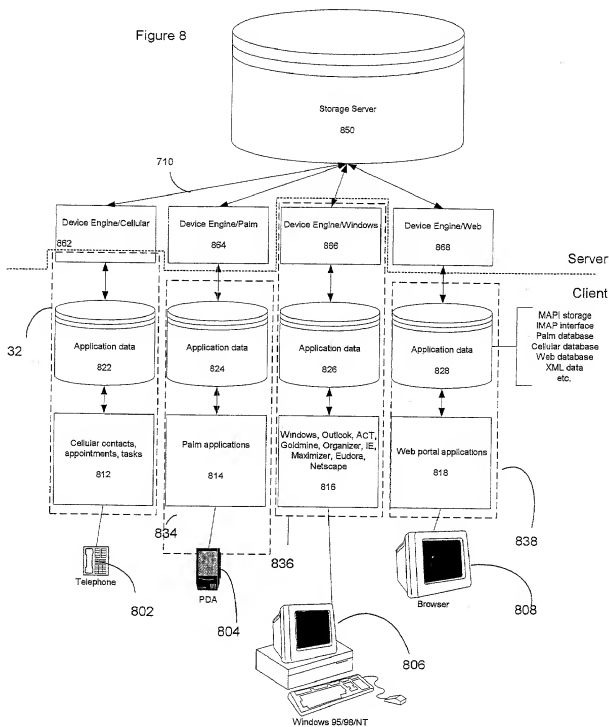


Figure 9A
Desktop
Device Engine

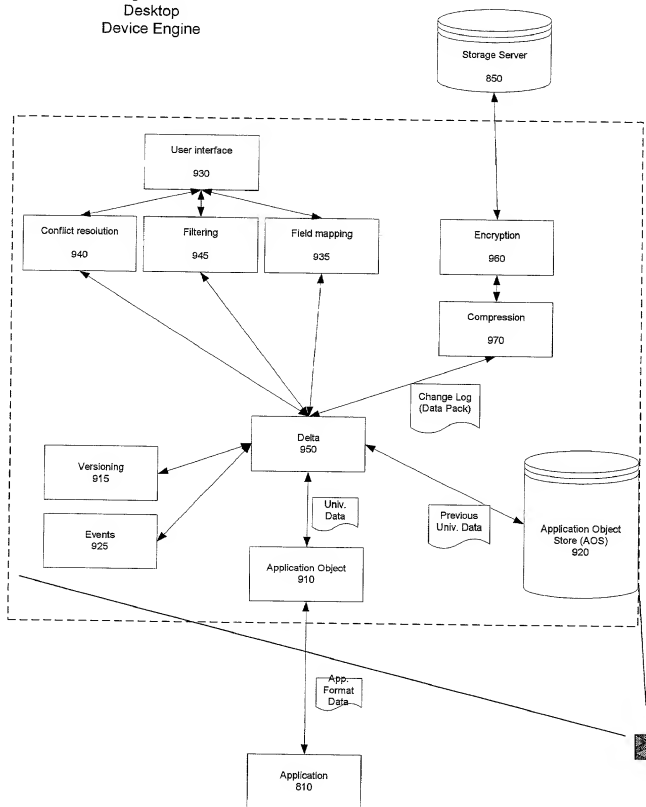
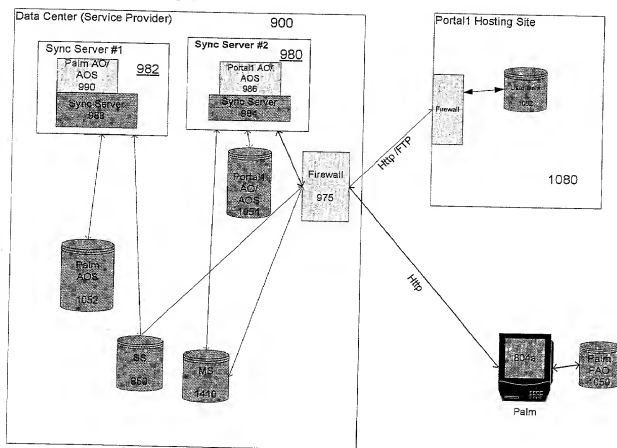


Figure 9B



Device Engine/Windows

Figure 10

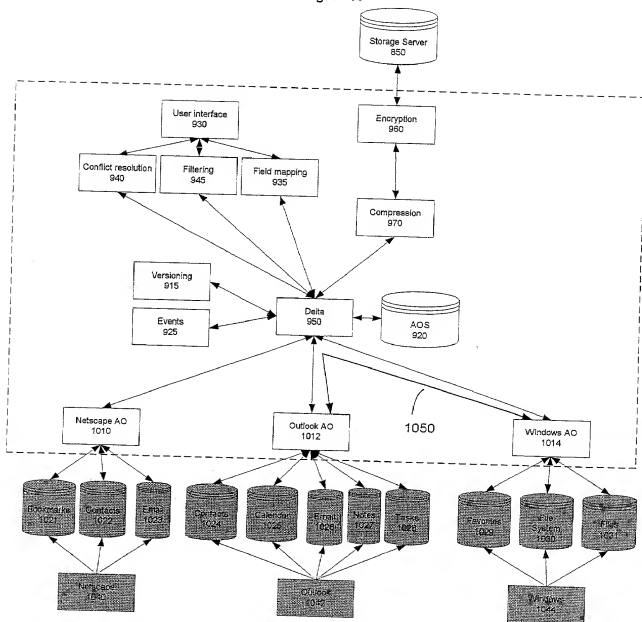


Figure 11

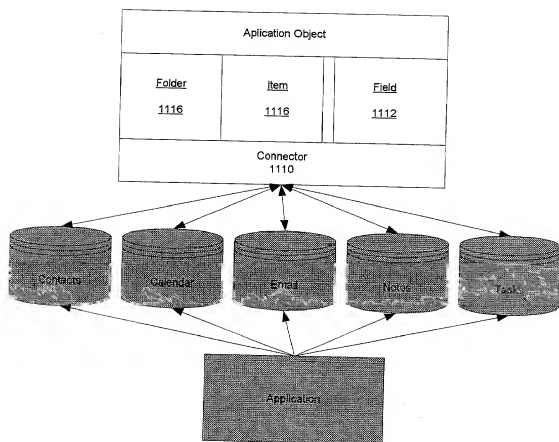
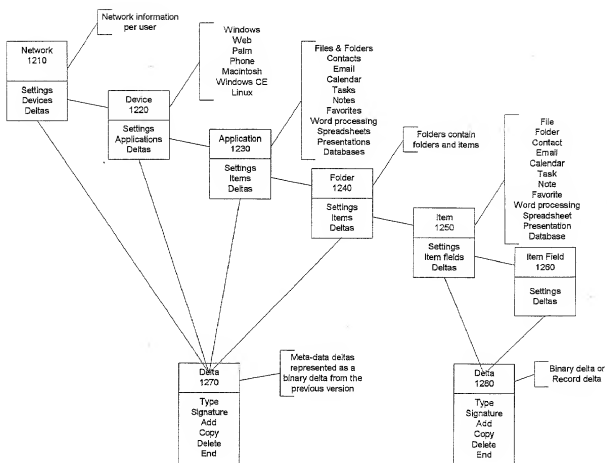


Figure 12
Object Hierarchy



[illegible]

Note Item 1310	Email Item 1320	Task Item 1330	Calendar Item 1340	Bookmark Item 1350	File Item 1360	Contact Item 1390
Note Object Categories Color Created Do Not AutoArchive Icon In Folder Message Class Modified Outlook Internal Version Outlook Version Read Size Subject	Account Attachment Bcc Billing Information Categories Cc Changed By Conversation Created Deferred Do Not AutoArchive Download State Due By Expires Flag Status Follow Up Flag From Have Replies Sent To Icon Importance In Folder Junk E-mail Type Message Class Mileage Modified Outlook Internal Version Outlook Version Read Received Remote Status Retrieval Time Sensitivity Sent Size Subject To Tracking Status	% Complete Actual Work Assigned Attachment Billing Information Categories Company Complete Contacts Conversation Created Date Completed Do Not AutoArchive Due Date Icon In Folder Message Class Mileage Modified Outlook Internal Version Outlook Version Owner Priority Read Recurring Reminder Reminder Override Default Reminder Sound Reminder Sound File Reminder Time Request Status Requested By Role Schedule+ Priority Sensitivity Size Start Date Status Subject Team Task To Total Work	All Day Event Attachment Billing Information Categories Conversation Created Do Not AutoArchive Duration End Icon Importance In Folder Location Meeting Status Message Class Mileage Modified Optional Attendees Organizer Outlook Internal Version Outlook Version Read Recurrence Recurrence Pattern Recurrence Range End Recurrence Range Start Recurring Remind Beforehand Reminder Reminder Override Default Reminder Sound Reminder Sound File Required Attendees Resources Response Requested Sensitivity Show Time As Size Start Subject	Name URL Created Modified Icon In Folder Version Size Channel Item 1370 Name URL Created Modified Icon In Folder Version Size	Name Type Version Size Created Modified Size Accessed Permissions Parent ID Attributes Icon Sharing Folder Item 1380 Name Type Version Size Created Modified Size Accessed Permissions Parent ID Attributes Icon Sharing	Account Address Selected Address Selected Assigned Name Assigned Name Assigned Phone Assigned Business Address Business Address Business Address City Business Address Country Business Address PO Box Business Address Postal Business Address Zone Business Phone Business Phone Page Business Phone 2 Business Phone 3 Business Phone 4 Business Phone 5 Business Phone 6 Business Phone 7 Business Phone 8 Business Phone 9 Business Phone 10 Business Phone 11 Business Phone 12 Business Phone 13 Business Phone 14 Business Phone 15 Business Phone 16 Business Phone 17 Business Phone 18 Business Phone 19 Business Phone 20 Business Phone 21 Business Phone 22 Business Phone 23 Business Phone 24 Business Phone 25 Business Phone 26 Business Phone 27 Business Phone 28 Business Phone 29 Business Phone 30 Business Phone 31 Business Phone 32 Business Phone 33 Business Phone 34 Business Phone 35 Business Phone 36 Business Phone 37 Business Phone 38 Business Phone 39 Business Phone 40 Business Phone 41 Business Phone 42 Business Phone 43 Business Phone 44 Business Phone 45 Business Phone 46 Business Phone 47 Business Phone 48 Business Phone 49 Business Phone 50 Business Phone 51 Business Phone 52 Business Phone 53 Business Phone 54 Business Phone 55 Business Phone 56 Business Phone 57 Business Phone 58 Business Phone 59 Business Phone 60 Business Phone 61 Business Phone 62 Business Phone 63 Business Phone 64 Business Phone 65 Business Phone 66 Business Phone 67 Business Phone 68 Business Phone 69 Business Phone 70 Business Phone 71 Business Phone 72 Business Phone 73 Business Phone 74 Business Phone 75 Business Phone 76 Business Phone 77 Business Phone 78 Business Phone 79 Business Phone 80 Business Phone 81 Business Phone 82 Business Phone 83 Business Phone 84 Business Phone 85 Business Phone 86 Business Phone 87 Business Phone 88 Business Phone 89 Business Phone 90 Business Phone 91 Business Phone 92 Business Phone 93 Business Phone 94 Business Phone 95 Business Phone 96 Business Phone 97 Business Phone 98 Business Phone 99 Business Phone 100 Business Phone 101 Business Phone 102 Business Phone 103 Business Phone 104 Business Phone 105 Business Phone 106 Business Phone 107 Business Phone 108 Business Phone 109 Business Phone 110 Business Phone 111 Business Phone 112 Business Phone 113 Business Phone 114 Business Phone 115 Business Phone 116 Business Phone 117 Business Phone 118 Business Phone 119 Business Phone 120 Business Phone 121 Business Phone 122 Business Phone 123 Business Phone 124 Business Phone 125 Business Phone 126 Business Phone 127 Business Phone 128 Business Phone 129 Business Phone 130 Business Phone 131 Business Phone 132 Business Phone 133 Business Phone 134 Business Phone 135 Business Phone 136 Business Phone 137 Business Phone 138 Business Phone 139 Business Phone 140 Business Phone 141 Business Phone 142 Business Phone 143 Business Phone 144 Business Phone 145 Business Phone 146 Business Phone 147 Business Phone 148 Business Phone 149 Business Phone 150 Business Phone 151 Business Phone 152 Business Phone 153 Business Phone 154 Business Phone 155 Business Phone 156 Business Phone 157 Business Phone 158 Business Phone 159 Business Phone 160 Business Phone 161 Business Phone 162 Business Phone 163 Business Phone 164 Business Phone 165 Business Phone 166 Business Phone 167 Business Phone 168 Business Phone 169 Business Phone 170 Business Phone 171 Business Phone 172 Business Phone 173 Business Phone 174 Business Phone 175 Business Phone 176 Business Phone 177 Business Phone 178 Business Phone 179 Business Phone 180 Business Phone 181 Business Phone 182 Business Phone 183 Business Phone 184 Business Phone 185 Business Phone 186 Business Phone 187 Business Phone 188 Business Phone 189 Business Phone 190 Business Phone 191 Business Phone 192 Business Phone 193 Business Phone 194 Business Phone 195 Business Phone 196 Business Phone 197 Business Phone 198 Business Phone 199 Business Phone 200 Business Phone 201 Business Phone 202 Business Phone 203 Business Phone 204 Business Phone 205 Business Phone 206 Business Phone 207 Business Phone 208 Business Phone 209 Business Phone 210 Business Phone 211 Business Phone 212 Business Phone 213 Business Phone 214 Business Phone 215 Business Phone 216 Business Phone 217 Business Phone 218 Business Phone 219 Business Phone 220 Business Phone 221 Business Phone 222 Business Phone 223 Business Phone 224 Business Phone 225 Business Phone 226 Business Phone 227 Business Phone 228 Business Phone 229 Business Phone 230 Business Phone 231 Business Phone 232 Business Phone 233 Business Phone 234 Business Phone 235 Business Phone 236 Business Phone 237 Business Phone 238 Business Phone 239 Business Phone 240 Business Phone 241 Business Phone 242 Business Phone 243 Business Phone 244 Business Phone 245 Business Phone 246 Business Phone 247 Business Phone 248 Business Phone 249 Business Phone 250 Business Phone 251 Business Phone 252 Business Phone 253 Business Phone 254 Business Phone 255 Business Phone 256 Business Phone 257 Business Phone 258 Business Phone 259 Business Phone 260 Business Phone 261 Business Phone 262 Business Phone 263 Business Phone 264 Business Phone 265 Business Phone 266 Business Phone 267 Business Phone 268 Business Phone 269 Business Phone 270 Business Phone 271 Business Phone 272 Business Phone 273 Business Phone 274 Business Phone 275 Business Phone 276 Business Phone 277 Business Phone 278 Business Phone 279 Business Phone 280 Business Phone 281 Business Phone 282 Business Phone 283 Business Phone 284 Business Phone 285 Business Phone 286 Business Phone 287 Business Phone 288 Business Phone 289 Business Phone 290 Business Phone 291 Business Phone 292 Business Phone 293 Business Phone 294 Business Phone 295 Business Phone 296 Business Phone 297 Business Phone 298 Business Phone 299 Business Phone 300 Business Phone 301 Business Phone 302 Business Phone 303 Business Phone 304 Business Phone 305 Business Phone 306 Business Phone 307 Business Phone 308 Business Phone 309 Business Phone 310 Business Phone 311 Business Phone 312 Business Phone 313 Business Phone 314 Business Phone 315 Business Phone 316 Business Phone 317 Business Phone 318 Business Phone 319 Business Phone 320 Business Phone 321 Business Phone 322 Business Phone 323 Business Phone 324 Business Phone 325 Business Phone 326 Business Phone 327 Business Phone 328 Business Phone 329 Business Phone 330 Business Phone 331 Business Phone 332 Business Phone 333 Business Phone 334 Business Phone 335 Business Phone 336 Business Phone 337 Business Phone 338 Business Phone 339 Business Phone 340 Business Phone 341 Business Phone 342 Business Phone 343 Business Phone 344 Business Phone 345 Business Phone 346 Business Phone 347 Business Phone 348 Business Phone 349 Business Phone 350 Business Phone 351 Business Phone 352 Business Phone 353 Business Phone 354 Business Phone 355 Business Phone 356 Business Phone 357 Business Phone 358 Business Phone 359 Business Phone 360 Business Phone 361 Business Phone 362 Business Phone 363 Business Phone 364 Business Phone 365 Business Phone 366 Business Phone 367 Business Phone 368 Business Phone 369 Business Phone 370 Business Phone 371 Business Phone 372 Business Phone 373 Business Phone 374 Business Phone 375 Business Phone 376 Business Phone 377 Business Phone 378 Business Phone 379 Business Phone 380 Business Phone 381 Business Phone 382 Business Phone 383 Business Phone 384 Business Phone 385 Business Phone 386 Business Phone 387 Business Phone 388 Business Phone 389 Business Phone 390 Business Phone 391 Business Phone 392 Business Phone 393 Business Phone 394 Business Phone 395 Business Phone 396 Business Phone 397 Business Phone 398 Business Phone 399 Business Phone 400 Business Phone 401 Business Phone 402 Business Phone 403 Business

Figure 14

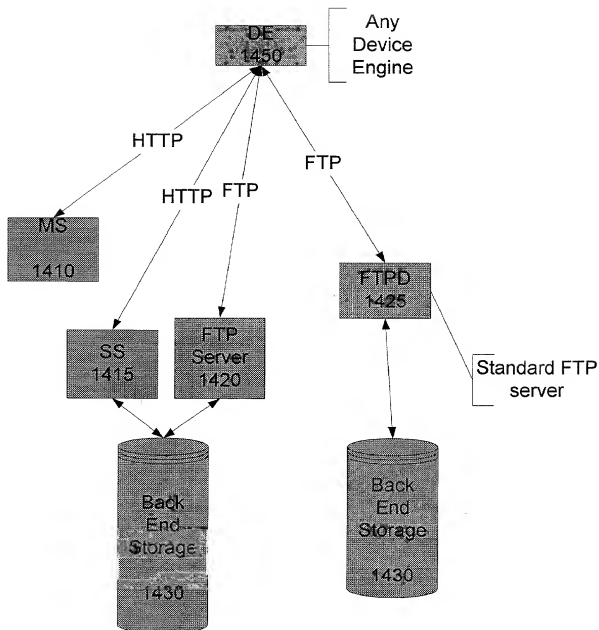


Figure 15
Pull Synchronization

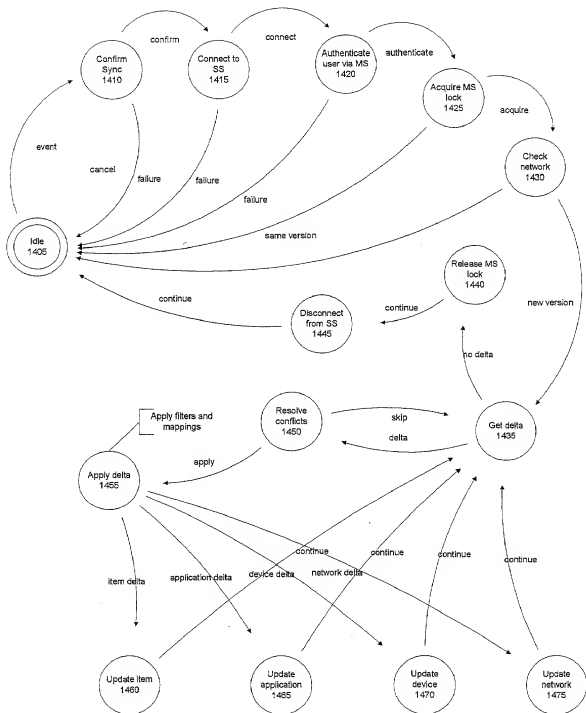


Figure 16
Push Synchronization

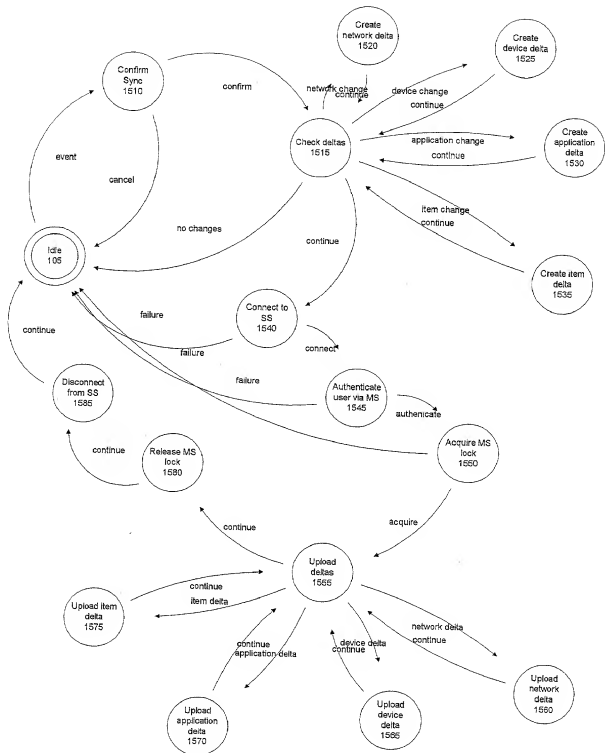


Figure
17

